The Pragmatic Programmers

TURING 图灵程序设计丛书

Python 数据科学入门

快速掌握数据采集与清洗、数据分析、
机器学习等数据科学领域常见任务和工具，
用Python轻松解决数据科学问题



[俄] Dmitry Zinoviev 著
熊子源 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

Dmitry Zinoviev

计算机科学教授，自2001年起一直在萨福克大学任教。研究兴趣包括计算机模拟与建模、网络科学、社交网络分析以及数字人文。拥有莫斯科国立大学物理学硕士学位和纽约州立大学石溪分校计算机科学博士学位。



熊子源

博士，毕业于国防科技大学电子科学与工程学院。读博期间开始接触Python，并在课题研究和项目研发中大量使用该语言。目前在北京某研究院工作，研究兴趣包括信号处理、机器学习、数据分析等。

TURING 图灵程序设计丛书

Python 数据科学入门

Data Science Essentials in Python



[俄] Dmitry Zinoviev 著

熊子源 译

人民邮电出版社

北京

图书在版编目（CIP）数据

Python数据科学入门 / （俄罗斯）德米特里·齐诺维耶夫（Dmitry Zinoviev）著；熊子源译. -- 北京：人民邮电出版社，2017.11
（图灵程序设计丛书）
ISBN 978-7-115-47060-7

Ⅰ. ①P… Ⅱ. ①德… ②熊… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2017)第253125号

内 容 提 要

本书以 Python 语言讲解数据科学基础知识，涵盖了数据采集、清洗、存储、检索、转换、可视化、高级数据分析（网络分析）、统计和机器学习等内容。具体内容包括：数据科学的 Python 核心特性，文本数据、数据库、表格形式的数值数据、series 和 frame、网络数据的使用，数据的绘制，概率与统计，机器学习。

本书面向研究生和本科生、数据科学教员、刚入门的数据科学专业人员，以及那些想拥有一本参考手册来帮助记住所有 Python 函数及参数的开发人员。

◆ 著　[俄] Dmitry Zinoviev
译　熊子源
责任编辑　岳新欣
执行编辑　吴威娜
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：10
字数：237千字　2017年 11 月第 1 版
印数：1－4 000册　2017年 11 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-6716号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

版权声明

献给我集美丽与智慧于一身的妻子安娜。献给我们的孩子们：优雅的芭蕾舞演员尤金妮亚和浪漫的游戏玩家罗曼。也献给2015年夏天我的第一门数据科学课。

我现在必须给你一个小小的科学指引，来扰乱你的思路。

——英国小说家 Marie Corelli

前　　言

2015年夏天，我在位于美国波士顿的萨福克大学使用Python教授数据科学入门课程，授课对象是一组经过选拔的本科生，本书的创作灵感正来源于这门课程。该课程是两个系列课程中的第一门课程，重点是数据的获取、清洗、组织和可视化，涉及统计学、机器学习和网络分析等相关内容。

数据的处理涉及庞大的体系和众多的Python模块（例如数据库、自然语言处理框架、JSON和HTML解析器，以及高性能数值数据结构，等等）。我很快意识到，不仅是本科生，甚至是经验丰富的专业人士，也很容易被这些浩瀚的知识所淹没。事实上，不得不承认，与我熟悉的领域相比，在进行数据科学和网络分析领域的研究时，我需要花更多时间去使用`help()`函数和浏览大量Python网络论坛。另外，我有时在课堂上会因为想不起某个函数名或可选参数而尴尬不已。

作为课程的一部分，我针对多类主题编辑了一套极具参考价值的备忘单。这些备忘单最终演变成了这本书。希望本书能够使你从大量函数名和可选参数中解脱出来，专注于数据科学和数据分析本身。

关于本书

本书涵盖了数据采集、清洗、存储、检索、转换、可视化、高级数据分析（网络分析）、统计和机器学习等内容。本书不是数据科学的综述或参考手册，不过你也能在第1章（“什么是数据科学”）找到如何开展数据科学的简要概述。阅读本书需要的先修知识包括数据科学的相关方法、统计学等。

第2章总结了Python数据结构，字符串、文件和与Web相关的函数，正则表达式，以及列表推导式。总结并非用于讲授这些知识，而是供你温习相关知识点。掌握Python对于一个成功的数据科学家而言无疑是非常重要的，你可以找到许多优秀的图书，进一步学习这门语言。

本书的第一部分介绍了如何使用不同类型的文本数据，包括处理结构化和非结构化的文本，使用NumPy和Pandas模块处理数值数据，以及网络分析。还有三章涉及数据分析的三个方面：使

用关系型和非关系型数据库、数据可视化以及简单的预测分析。

本书是一本半叙述半参考性的书。你可以直接按顺序阅读，也可以先找出你关心的函数或概念，然后查阅相关的说明和示例。若是按顺序阅读，而你又有一定的Python编程经验，就可以直接跳过第2章（“数据科学的Python核心”）。如果你不打算使用外部数据库（比如MySQL），也可以忽略第4章（“使用数据库”）。最后，如果你对统计学已经有一定了解，那么完全可以跳过第9章（“概率与统计”）的前两个单元，直接阅读第47单元（“以Python的方式完成统计”）。

关于读者

你可以在此了解自己是否需要本书。

本书面向研究生和本科生、数据科学教员、刚入门的数据科学专业人员（特别是从R语言转为使用Python的人），以及那些想拥有一本参考手册来帮助记住所有Python函数及参数的开发人员。

如果你是他们中的一员，那就不要犹豫，直接开始阅读吧。

关于软件

尽管在是否要从Python 2.7转到Python 3.3或者更高版本这个问题上，目前尚存在一定争论，但我还是支持使用新版本的Python。许多新的Python软件是针对Python 3.3开发的，而且多数遗留软件都已经成功移植到Python 3.3。考虑到这种趋势，选择将过时的Python 2.7恐怕并非明智之举，即使它现在还很流行[1]。

本书所有Python示例需要用到的模块都列在了下表中。

表1　本书使用的软件组件

包	使用的版本	包	使用的版本
BeautifulSoup	4.3.2	community	0.3
json	2.0.9	html5lib	0.999
matplotlib	1.4.3	networkx	1.10.0
nltk	3.1.0	numpy	1.10.1
pandas	0.17.0	pymongo	3.0.2
pymysql	0.6.2	python	3.4.3
scikit-learn	0.16.1	scipy	0.16.0

① Python 2.7是2.x系列的最后一个版本，支持时间到2020年。——译者注

在这些模块中，除了需要单独安装的community版模块[①]和Python解释器外，其他都已经包含在Anaconda发行版中。Anaconda发行版是由Continuum Analytics公司开发的免费软件[②]。

如果你想试用一下数据库（或者你的工作就要用到数据库），那么你还需要下载并安装MySQL[③]和MongoDB[④]这两个数据库。它们都是免费的，能运行在Linux、Mac OS和Windows平台上。

关于引号

Python允许用户使用以下方式表示字符串：'单引号'、"双引号"、'''三个单引号'''，甚至是"""三个双引号"""（其中后两个可表示多行字符串）。然而，不论程序中使用哪种引号，当打印字符串时，通常都使用单引号。

许多其他语言（C、C++、Java）会在不同场合使用单引号和双引号：单引号用于单个字符，双引号用于字符串。本书沿用这种区分方式——对单个字符使用单引号，对字符串使用双引号。

关于本书的论坛

本书的社区论坛可在Pragmatic Programmers网站上找到[⑤]。你可以在论坛中提问、发表评论和提交勘误。

另一个很好的问答资源（不限于本书）是在Stack Exchange网站上新创建的数据科学板块[⑥]。

轮到你了

每章最后都有一个叫作“轮到你了”的单元。这个单元描述了几个项目，你可以自己独立（或与你信任的人一起）完成这些项目，以加强对本书内容的理解。

单个星号（*）标记的项目是最简单的。完成这些项目只需要前面章节中提到的函数方面的知识。预计不超过三十分钟就可以完成“单星项目”。你可以在附录2（“单星项目的解决方案”）中找到参考的解决方法。

两个星号（**）标记的是较难的项目。完成这些项目可能需要一小时甚至更长的时间，当然这也取决于你的编程技能和习惯。“两星项目”需要使用某些中级数据结构和周密的算法。

① pypi.python.org/pypi/python-louvain/0.3
② www.continuum.io
③ www.mysql.com
④ www.mongodb.com
⑤ pragprog.com/book/dzpyds
⑥ datascience.stackexchange.com

最后，标记为三个星号（***）的项目是最难的。一些“三星项目”甚至可能没有一个完美的解决方案，因此如果你解不出来也不必沮丧。不过，通过练习“三星项目”，你一定可以成为更出色的程序员和更优秀的数据科学家。而且如果你是教育工作者，可以考虑将“三星项目”作为期中作业。

现在，让我们开始吧！

Dmitry Zinoviev
dzinoviev@gmail.com
2016年8月

致　　谢

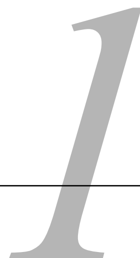
衷心感谢Xinxin Jiang教授（萨福克大学）对本书统计部分所提出的宝贵意见。同时感谢Jason Montojo（*Practical Programming: An Introduction to Computer Science Using Python 3*的作者之一）、Amirali Sanatinia（东北大学）、Peter Hampton（阿尔斯特大学）、Anuja Kelkar（卡内基梅隆大学）和Lokesh Kumar Makani（Skyhigh Networks公司），本书的问世离不开他们的宝贵意见。

目　　录

学无止境。

——俄罗斯作家Kozma Prutkov

第1章 什么是数据科学



相信你对数据科学已经有了一些了解，不过我们还是可以一起来回顾一下！数据科学是从数据中提取知识的学科。它依赖于计算机科学（数据结构、算法、可视化、大数据支持和通用编程）、统计学（回归和推理），以及领域知识（用于提问和解释结果）。

传统意义上的数据科学涵盖多种不同主题，有些是你可能已经熟悉的，而有些是你将在本书中遇到的。

- **数据库**，提供信息的存储和集成。关于关系型数据库和文档存储的信息请参见第4章。
- **文本分析**和**自然语言处理**，让我们可以通过将定性文本转化成定量变量，实现"用文字计算"。你是否对情感分析工具感兴趣？那么阅读本书的第16单元再合适不过了。
- **数值数据分析**和**数据挖掘**，可搜索出变量之间的不变性和相互关系。这是第5章和第6章的主题。
- **复杂网络分析**，其实并不复杂。所谓复杂网络，是指任意互连实体的集合。第7章介绍了如何将复杂网络分析简单化。
- **数据可视化**，不仅富有美感，而且非常实用，尤其是当你想说服数据赞助商再次提供赞助的时候。如果说一图胜千言，那么第8章的价值就远超过本书的其他部分。
- **机器学习**（包括聚类、决策树、分类和神经网络），试图让计算机学会"思考"，并根据样本数据进行预测。第10章对如何实现这样的功能进行了说明。
- **时间序列处理**，或者更一般地说，**数字信号处理**，是股市分析师、经济学家以及音频和视频领域的研究人员不可或缺的工具。
- **大数据分析**，通常指对频繁生成和获取的大于1TB的非结构化数据（文本、音频、视频）进行分析。大数据如此之大，以至于难以在本书中进行完整的介绍。

不论针对哪种分析类型，数据科学首先是科学，然后才是魔法。因此，它是一个严格遵循以数据采集为起点、以结果报告为终点的基本处理过程。在本章中，你将了解数据科学的基本过程，

包括：常见数据分析研究的步骤、数据的获取来源，以及常见项目报告的结构。

第 1 单元 数据分析步骤

常见的数据分析研究步骤通常与一般的科学发现顺序一致。

数据科学发现从要解决的问题和要应用的分析方式开始。最简单的分析类型是**描述性**的，通常使用一种可视化的形式给出数据集总量的描述。不论你接下来打算做什么，至少需要描述一下数据！在**探索性**数据分析的过程中，你需要尝试找出现有变量之间的相互关系。基于统计的**推断分析**可以帮助你利用手头上少量的数据样本，对更大的群体进行描述。**预测分析**是从过去的规律中预测未来。**因果分析**能找出相互影响的变量。最后，**机制性**数据分析准确揭示了一个变量如何影响另一个变量。

然而，分析结果的好坏依赖于数据的质量，因此引出了如下问题：什么样的数据集是理想的呢？在理想情况下，什么样的数据能够解决问题呢？另外，理想的数据集可能根本就不存在，或者是很难甚至不可能获取。对于这种情况，一个较小的或者特征不那么丰富的数据集还依然有用吗？

幸运的是，从Web或数据库获取原始数据并不难，有大量基于Python的工具可用于下载和解析这些数据。你可以在第2单元（“数据获取途径”）中进一步了解这些工具。

应该注意到，完美的数据是不存在的。难免会遇到有丢失值、异常值和其他“非标准”项的“脏”数据。“脏”数据的例子包括：未来的出生日期、负年龄和负体重，以及不合理的电子邮件地址（noreply@）。因此，一旦获得了原始数据，接下来就是使用数据清洗工具和统计知识来正则化数据集。

完成上述处理后，就可以使用干净的数据，开展描述性和探索性分析。这一步的成果通常包括散点图（参考第44单元）、直方图和统计总结（参考第46单元）。它们赋予了你对数据独有的感觉——这是一种在后续研究中不可或缺的对数据集（尤其是针对多维数据集）的直观认识。

现在离实现预测只有一步之遥了。你手中的数据模型工具，在经过恰当的训练后，就可以实现预测功能。值得注意的是，不能忽视对模型的质量及其预测精度的评估！

至此，你可以摘掉统计学家和程序员的帽子，换上一顶领域专家的帽子了。你已经得到了一些成果，但它们称得上是领域内的重要成果吗？换句话说，是否有人关心这些成果，还有，这些成果带来了什么不一样的认知？设想一下，你被聘用为一名评论员，来评价自己的工作。你做的哪些是正确的，哪些是错误的？如果再给你一次机会，哪些工作你能做得更好或者不同？你是否

会使用不同的数据，作出不同类型的分析，提出不同的问题，抑或建立一个不同的模型？一定有人会提出这些问题。提前进行思考，对你是大有裨益的。当你还沉浸在这些字里行间时，寻觅答案的征程已然开始。

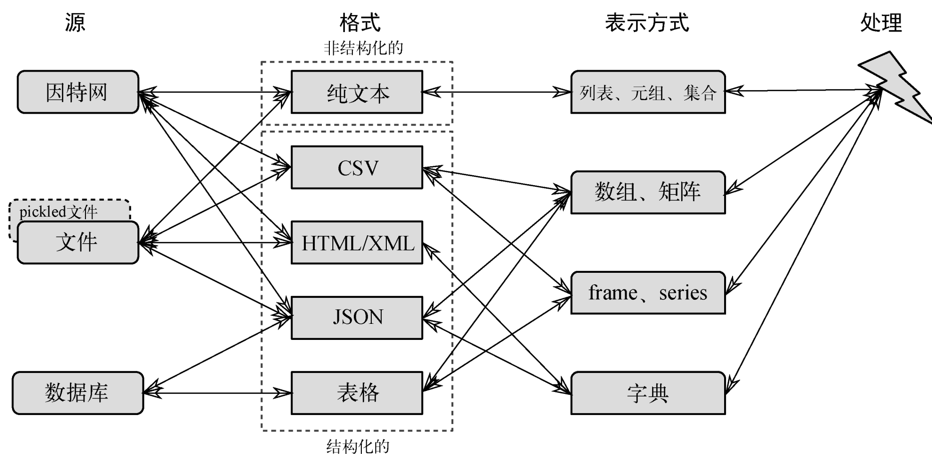
最后，你必须完成一个报告，说明你处理数据的方式及理由、建立了什么模型、可能得出什么结论、可能作出什么预测。本章末尾（第3单元）讲解了报告的结构。

作为一本数据科学领域的Python手册，本书的重点是典型数据分析步骤中早期的、最随意，同时也是最有创意的部分：数据的获取、清洗、组织和分级。本书几乎不涉及数据建模的内容，包括预测数据的建模。（当然，完全抛开数据建模是不合理的，毕竟这是魔法的真正所在！）一般来说，结果解释、质疑和报告非常依赖于特定的领域，这些内容可在专门的教材中找到。

第 2 单元

数据获取途径

数据获取涉及获得包含来自各种输入器件的数据源、从器件中提取数据，以及将其转换为适于进一步处理的表示方式，如下图所示。



数据的三个主要来源是因特网（即万维网）、数据库，以及本地文件（可能是先前手动下载或利用其他软件下载得到的）。某些本地文件可能是通过Python程序生成的，包括序列化的或“pickled”的数据（更详细的解释请参考第12单元）。

来自器件的数据格式多种多样。在后续章节中，你将接触到最流行的数据格式及其对应的数据分析方式和方法。

- 自然语言中的非结构化纯文本（比如英语、汉语）
- 结构化数据，包括：
  - 逗号分隔值（CSV）文件中的表格数据
  - 数据库中的表格数据
  - 使用超文本标记语言（HTML）或更一般的可扩展标记语言（XML）的标记数据
  - JavaScript对象表示法（JSON）中的标记数据

根据所提取数据的原始结构，以及进一步处理的目的和性质，本书示例中的数据均表示为原生的Python数据结构（列表和字典），或支持特定操作的高级数据结构（numpy中的数组和pandas中的frame数据）。

我将尽可能地呈现一个完全自动化的数据处理流程（获取、清洗和变换原始数据；描述性和探索性数据分析；数据建模和预测）。为此，我避免使用交互式GUI工具，因为GUI的处理方式很少能脚本化以实现批处理模式，且几乎不记录任何处理历史。为了提高模块化程度、可重用性和可恢复性，我会把较长的流程分解为较短的子流程，并将中间结果保存到Pickle（参考第12单元）或JSON（参考第15单元）文件中。

自动化流程自然产生了可重用的代码：一组任何人都可以执行的Python脚本。这些脚本可以将原始数据转换为报告中描述的最终结果；在理想情况下，这一过程不需要任何额外的人机交互。其他研究人员能使用可重用的代码对模型和结果进行验证，并应用你开发的程序解决他们遇到的问题。

第 3 单元 报告的结构

项目报告是我们（数据科学家）向数据赞助商（客户）提交的。报告通常包括以下内容：

- 摘要（对项目的一个言简意赅的描述）
- 引言
- 所使用的数据获取及处理方法
- 所获得的结果（不包含中间的和不重要的结果，应把这些内容放到附录中）
- 结论
- 附录

除了不是特别重要的结果和图形外，附录应包含用于处理数据的所有可重用的代码：可在没有命令行参数和用户交互情况下执行的、具有良好注释的脚本。

提交的最后一部分是原始数据：以可复现的方式执行代码所需的所有数据文件，除了由数据赞助者提供且未被修改的文件以外。README文件通常用于解释数据的来源和每个附加数据文件的格式。

上述的结构仅作为项目结构的建议，并非一成不变。根据数据赞助商的提议以及所达成的共识，也许会给出其他的结构。

轮到你了

作为介绍性的一章，你能从本章了解到数据科学的基本过程：常见数据分析研究的步骤，获取数据的方式和各种数据格式，以及常见项目报告的结构。本书其余部分介绍了在Python中对基本的数据科学来说至关重要的特征，以及为复杂度适中的数据科学项目提供算法和统计支持的各种Python模块。

在继续阅读之前，我们通过一个简单的项目来热热身。计算机程序员有一个良好的传统，那就是通过编写一个输出“Hello，World！”的程序，将初学者引入新的编程语言。我们应该遵循这个传统。

- Hello, World!*

  编写一个在Python命令行输出“Hello, World!”的程序。

我用了所有我会的语言与他们沟通，包括多少懂一点的高地德语、低地德语、拉丁语、法语、西班牙语、意大利语和通用语，可惜都没有用。

——英裔爱尔兰讽刺作家Jonathan Swift

第2章 数据科学的Python核心

Python核心的一些特性对于数据分析而言至关重要。本章将给出最为重要的特性：字符串函数、数据结构、列表推导式、计数器、文件和Web函数、正则表达式、globbing以及pickling数据。通过学习本章，你将掌握如何使用Python从本地磁盘文件和网络中提取数据，用恰当的数据结构存储数据，定位与给定模式相匹配的位和片段，以及为方便后续处理而对Python对象进行序列化和反序列化操作。这些功能并非数据科学或数据分析任务所特有的，在许多其他应用中也会用到。

人们往往错误地认为，高级编程工具的出现使得低级编程过时了。毕竟独立于Python的Anaconda发行版就提供了350多个Python包，谁会需要自己去实现拆分字符串和打开文件这样低级的函数呢？但是不可否认，世界上依然存在大量非标准的数据源，处理这些数据就必须编写低级的函数。

所有的标准数据frame、series、CSV读取器和文字分词器，都遵循其创建者设定的规则。一旦遇到任何违反规则的情况，这些工具都将束手无策。此时，你就应该摘去数据科学家的光环，吹去这本书上的灰尘，做回一个谦卑而务实的程序员。

为了接点“地气”，你可能需要了解字符串函数——相关的内容就在第4单元中。

第4单元 理解基本的字符串函数

字符串是计算机世界和人类世界之间的基本交互单元。最初，几乎所有的原始数据都存储为字符串。在本单元中，你将学习如何评估和操作文本字符串。

本单元中描述的所有函数都是内置的str类的成员函数。

大小写转换函数返回原始字符串s的一个副本：lower()函数将所有字符转换为小写；upper()函数将所有字符转换为大写；capitalize()函数将第一个字符转换为大写，同时将其他所有字符转换为小写。这些函数不会影响非字母字符。大小写转换函数是规范化的一个重要元素，第16单元讲解了规范化的相关内容。

判定（predict）函数根据字符串s是否属于适当的类而返回True或False：islower()函数检查所有字母字符是否为小写；isupper()函数检查所有字母字符是否为大写；isspace()函数检查所有字符是否为空格；isdigit()函数检查所有字符是否为范围0 ~ 9中的十进制数字；isalpha()函数检查所有字符是否为a ~ z或A ~ Z范围内的字母字符。使用这些函数，你可以识别有效的单词、非负整数、标点符号等。

Python有时会将字符串数据表示为原始的二进制数组，而非字符串，尤其是当数据来自外部源（外部文件、数据库或Web）时。Python使用符号b来标识二进制数组。例如，bin = b"Hello"是一个二进制数组；s = "Hello"是一个字符串。s[0]和bin[0]分别是'H'和72，其中72是字符'H'的ASCII码。**解码**函数将二进制数组转换为字符串或反之：bin.decode()将二进制数组转换为字符串，而s.encode()将字符串转换为二进制数组。许多Python函数都需要将二进制数据转换为字符串，然后再做处理。

字符串处理的第一步是去除不需要的空白（包括换行符和制表符）。函数lstrip()（left strip）、rstrip()（right strip）和strip()分别在字符串的开始处、结束处或对整个字符串删除所有空格（不删除字符串内部空格）。经过这些删除操作后，得到的可能会是一个空字符串！

```
"    Hello, world! \t\t\n".strip()
⇒ 'Hello, world!'
```

字符串通常包含多个标记符，用空格、冒号和逗号这样的分隔符分割。函数split(delim='')使用delim作为分隔符，将字符串s分割为子字符串组成的一个列表。如果未指定分隔符，Python会使用空白字符来分割字符串，并将所有连续的空白合并：

```
"Hello,  world!".split() # 两个空格!
⇒ ['Hello,', 'world!']
"Hello,  world!".split(" ") # 两个空格!
⇒ ['Hello,', '', 'world!']
"www.networksciencelab.com".split(".")
⇒ ['www', 'networksciencelab', 'com']
```

连接函数join(ls)——分割函数的姐妹函数——将字符串列表ls连接在一起，形成一个字符串，并使用特定的对象字符串作为连接符。你可以使用join()函数重组字符串片段：

```
", ".join(["alpha", "bravo", "charlie", "delta"])
```

⇒ **'alpha, bravo, charlie, delta'**

在这个例子中，join()函数仅在字符串之间插入连接符，而在第一个字符串前或最后一个字符串后都不插入连接符。这种分割字符串和再次连接字符片段的操作，往往用于将已有的分隔符替换为给定的连接符，字符串本身看上去并没有明显的变化：

```
"-".join("1.617.305.1985".split("."))
```

⇒ **'1-617-305-1985'**

有时，为了从字符串中删除不需要的空白，你可能会同时使用这两个函数。当然你也可以使用基于正则表达式的替换来实现相同的效果（参考第10单元第2小节）。

```
" ".join("This string\n\r  has    many\t\tspaces".split())
```

⇒ **'This string has many spaces'**

函数find(needle)返回对象字符串中子字符串needle第一次出现的索引值，当子字符串不存在时，返回-1。该函数区分大小写，它可用于在字符串中查找感兴趣的片段（如果存在的话）。

```
"www.networksciencelab.com".find(".com")
```

⇒ **21**

函数count(needle)返回对象字符串中子字符串needle非重叠出现的次数，该函数也区分大小写。

```
"www.networksciencelab.com".count(".")
```

⇒ **2**

字符串是任何数据处理程序的重要组件，但它既不是唯一的组件，也不是最高效的组件。还可以使用列表、元组、集合和字典来捆绑字符串和数值数据，以实现高效的搜索和排序。

第 5 单元

选择合适的数据结构

列表、元组、集合和字典是Python中最常用的复合数据结构，它们都属于容器类的数据结构。

Python用数组的方式实现**列表**。列表的搜索时间是线性的，因此用列表来存储大量可搜索的数据是不切实际的。

元组是不可变的列表，创建后就无法再更改。元组的搜索时间也是线性的。

与列表和元组不同，**集合**不是序列：集合项不存在索引。集合最多只能存储一个项的副本，具有次线性$O(\log(N))$的搜索时间。集合非常适合于成员查找和消除重复项（如果将包含重复项的列表转换为集合，则重复项将会消失）：

```
myList = list(set(myList)) # 删除myList中的重复项
```

可以将列表数据转换为查询成员速度更快的集合数据。在下面的例子中，`bigList`是以十进制字符串表示的前1000万个整数的列表：

```
bigList = [str(i) for i in range(10000000)]
"abc" in bigList # 耗时0.2秒
bigSet = set(bigList)
"abc" in bigSet # 耗时15~30微秒——快了10 000倍！
```

字典构建了从键到值的映射。任何可哈希的数据类型（数字、布尔、字符串、元组）的对象都可以作为键，且同一字典中的不同键可以属于不同的数据类型。Python对字典值的数据类型也没有限制。字典具有次线性$O(\log(N))$搜索时间，它非常适合用于键值的查找。

你可以通过(键, 值)这样的元组列表创建字典，也可以使用内置类构造函数`enumerate(seq)`创建字典，这样得到的字典，其键为各项在`seq`中的序列号：

```
seq = ["alpha", "bravo", "charlie", "delta"]
dict(enumerate(seq))
```

⇒ **{0: 'alpha', 1: 'bravo', 2: 'charlie', 3: 'delta'}**

另一种从键序列（`kseq`）和值序列（`vsec`）创建字典变量的巧妙方法，是使用内置类构造函数`zip(kseq, vseq)`（两个序列必须具有相同的长度）：

```
kseq = "abcd" # 字符串也是一个序列
vseq = ["alpha", "bravo", "charlie", "delta"]
dict(zip(kseq, vseq))
```

⇒ **{'a': 'alpha', 'c': 'charlie', 'b': 'bravo', 'd': 'delta'}**

Python将`enumerate(seq)`和`zip(kseq, vseq)`（以及经典的`range()`函数）实现为列表生成器。列表生成器提供了一个迭代器接口，这使得我们可以在`for`循环中使用它们。与真正的列表不同的是，列表生成器只在需要时才生成下一个元素，这可以说是一种巧妙的偷懒方式。列表生成器便于处理大型列表，甚至允许“无限”的列表。你可以通过调用`list()`函数，将列表生成器显式强制转换为列表。

第 6 单元

通过列表推导式理解列表

列表推导式是一个将数据集（不一定是列表）转换为列表的表达式。通过列表推导式，可以

实现对所有或某些列表元素应用相同的操作，例如将所有元素转换为大写或每个元素的幂级数运算结果。

转换过程如下。

(1) 表达式遍历数据集并访问集合中的每一项。
(2) 为每一项计算可选的布尔表达式（默认值为True）。
(3) 如果布尔表达式为True，则计算当前项目的循环表达式，并将其值附加到结果列表中。
(4) 如果布尔表达式为False，则忽略该项。

这里给出一些比较简单的列表推导式：

```
# 复制myList；等同于myList.copy()或者myList[:]，但二者的效率都没有列表推导式高
[x for x in myList]
# 提取非负项
[x for x in myList if x >= 0]
# 用Mylist各项的平方构建一个新列表
[x**2 for x in myList]
# 用Mylist非零项的倒数构建一个新列表
[1/x for x in myList if x != 0]
# 从打开的infile文件中选出所有的非空行，
# 并删除这些行开头和结尾的空格
[l.strip() for l in infile if l.strip()]
```

在最后一个例子中，对于每个列表项，函数strip()被执行了两次。如果不想使用这种重复的表达方式，那么可以使用下面这样的嵌套列表推导式。其中，内部的列表推导式删除空白，外部的列表推导式消除空字符串：

```
[line for line in [l.strip() for l in infile] if line]
```

如果列表推导式被包含在圆括号中，而不是在方括号中，则程序将返回一个列表生成器对象：

```
(x**2 for x in myList) # 结果为：<generator object <genexpr> at 0x...>
```

列表推导式的结果通常是重复项目的列表：数字、单词、词干和标题。如果想知道哪个项目是最常见或最少见的，可以借助于Counter类（参考第7单元），它是一个用于收集这类统计数据的免费工具。

第 7 单元　使用计数器

计数器是一种字典式集合，用于给（另一个）集合项目计数。计数器定义在collections模块中。你可以将要计数的集合传递给构造函数Counter，然后使用函数most_common(n)来获取n个出现频率最高的项及对应频率的列表（如果没有提供参数n，则函数返回的将是一个针对所

有项目的列表）。

```
from collections import Counter
phrase = "a man a plan a canal panama"
cntr = Counter(phrase.split())
cntr.most_common()
```

⇛ **[('a', 3), ('canal', 1), ('panama', 1), ('plan', 1), ('man', 1)]**

为了便于查询，可将列表转换为字典：

```
cntrDict = dict(cntr.most_common())
```

⇛ **{'a': 3, 'canal': 1, 'panama': 1, 'plan': 1, 'man': 1}**

```
cntrDict['a']
```

⇛ **3**

你将在第35单元的第3小节（“唯一性、计数、会员资格”）中了解到基于pandas模块的、更为通用的计数工具。

第 8 单元

使用文件

文件是用于长期存储数据的非易失性容器。与文件相关的常见操作包括打开文件，从文件读取数据或将数据写入文件，以及关闭文件。你可以打开文件进行读取（默认模式，定义为"r"）、（覆盖）写入（"w"）或追加写入（"a"）。以写入方式打开文件，会在没有任何通知的情况下，破坏文件的原始内容；另外，试图打开一个不存在的文件进行读取操作会导致异常：

```
f = open(name, mode="r")
« read the file »
f.close()
```

Python给这种编程范式提供了一个有效的替代品：with语句。with语句允许显式地打开一个文件，同时保证在退出Python后能自动关闭文件，从而避免了跟踪那些已打开却不再需要的文件。

```
with open(name, mode="r") as f:
  « read the file »
```

一些模块，例如pickle模块（第12单元讨论了该模块的使用），要求以二进制模式（"rb"、"wb"或"ab"）打开文件。在读取或写入原始二进制数组时，同样需要使用二进制模式。以下函数能从已打开的文件f中读取文本数据：

```
f.read() # 以字符串或二进制的方式读入所有数据
f.read(n) # 以字符串或二进制的方式读入前n字节的数据
```

```
f.readline() # 以字符串的方式读入下一行
f.readlines() # 以字符串的方式读入所有行
```

根据实际需要，可以混合和搭配使用这些函数。例如，可以读取第一个字符串，接着读取下5个字节，然后再读下一行，最后读取文件的剩余部分。这些函数的返回结果都不会删除换行符。通常，在不确定文件是否较小的情况下，使用函数`read()`和`readlines()`是不安全的。

以下函数将文本数据写入已打开的文件`f`：

```
f.write(line) # 写字符串数据或二进制数据
f.writelines(ines) # 写字符串数据列表
```

这些函数不会在写入的字符串末尾添加换行符。如果需要，你得自己添加。

第 9 单元　上网

根据WorldWideWebSize网站[1]的统计结果，能被索引的Web至少包含48.5亿网页[2]。其中部分网页可能是我们感兴趣的。`urllib.request`模块包含从Web下载数据的函数。可以手动下载单个数据集，将其保存到缓存目录中，再使用Python脚本进行分析，不过这种方法并不可取。另外，某些数据分析项目需要自动化迭代或递归下载。

无论要从Web上获取什么，第一步都是用`urlopen(url)`函数打开网址，以获得打开网址的句柄。一旦打开了网址，对应的网址句柄就类似于以只读方式打开的文件句柄：可以使用函数`read()`、`readline()`和`readlines()`来访问数据。

鉴于Web和互联网的动态特性，无法打开网址的概率要高于无法打开本地文件的概率。因此，务必将任何与Web相关的函数调用都包含在一个异常处理的语句中：

```
import urllib.request
try:
  with urllib.request.urlopen("http://www.networksciencelab.com") as doc:
    html = doc.read()
    # 如果数据读取成功，连接就会自动关闭
except:
  print("Could not open %s" % doc, file=sys.err)
  # 不要假装已经读到了文件!
  # 一定要在这里执行错误处理程序
```

当数据集部署在需要身份验证的网站上时，就不能使用`urlopen()`函数了。此时，应使用提供安全套接层（SSL，例如OpenSSL）的模块。

① www.worldwidewebsize.com

② 网页数量的统计结果是动态变化的，在翻译此处时所查到的最新数据是47.2亿。——译者注

urllib.parse模块提供了用于解析和构建网址的友好工具。函数urlparse()将网址分成六个元素的元组：协议（比如http）、网络地址、文件系统路径、参数、查询和片段：

```
import urllib.parse
URL = "http://networksciencelab.com/index.html;param?foo=bar#content"
urllib.parse.urlparse(URL)

⇒ ParseResult(scheme='http', netloc='networksciencelab.com',
⇒   path='/index.html', params='param', query='foo=bar',
⇒   fragment='content')
```


函数urlunparse(parts)使用urlparse()的返回值parts构造有效的网址。如果在解析一个网址之后再将其重新构建，则结果可能与原始网址稍有不同，但功能上是完全等效的。

第 10 单元

使用正则表达式实现模式匹配

正则表达式是一种基于模式匹配的搜索、拆分及替换字符串的强大机制。re模块提供模式描述语言和用于匹配、搜索、拆分和替换字符串的函数。

从Python的角度来看，正则表达式只是一个包含模式描述的字符串。将一个需要多次使用的正则表达式编译为Pattern对象后，可以使模式匹配更加高效：

```
compiledPattern = re.compile(pattern, flags=0)
```

编译在不影响正确性的前提下，大大提高了模式匹配效率。可以根据需要，在编译时或之后执行时，设定模式匹配标志。最常见的标志是re.I（忽略字符大小写）和re.M（告诉re在多行模式下工作，并让运算符^和$也匹配行的开始或结束）。如果要组合使用多个匹配标志，只需直接添加即可。

理解正则表达式语言

部分正则表达式语言汇总在下表中。

表2　正则表达式语言

基本操作[1]	
.	除换行符之外的任意字符
a	字母a

① 在Python 3.x中，字符串默认都是采用Unicode编码，这对\w、\W、\b、\B、\d、\D、\s和\S会有些影响。详情请参考Python官方手册（https://docs.python.org/3.3/library/re.html?Highlight= re#re.ASCII）。——译者注

（续）

基本操作	
ab	字符串ab
x|y	x或y
\y	特殊字符y（例如^+{}$()[]|-?.*）的转义符
字符类	
[a-d]	a、b、c、d中的某个字符
[^a-d]	除了a、b、c、d外的一个字符
\d	一个数字字符
\D	一个非数字字符
\s	一个空白字符
\S	一个非空白字符
\w	一个字母数字字符①
\W	一个非字母数字字符
数量相关	
x*	0个或多个x
x+	1个或多个x
x?	0个或1个x
x{2}	2个且仅2个x
x{2,5}	2至5个x
转义字符	
\n	换行符
\r	回车符
\t	Tab
定界符	
^	字符串起始处
\b	单词边界
\B	非单词边界
$	字符串结尾
组	
(x)	捕获组
(?:x)	非捕获组

位于字符类表达式中间或结尾处的插入符（^）和连接符（-）不具有特殊含义，表示字符'^'和'-'。组可以改变操作的顺序。当成功匹配时，匹配捕获组的子字符串也包括在结果列表中。

① 对于Python 3.x默认的Unicode编码，\w指的是一个Unicode字符；而对于ASCII编码，\w指任意一个字母或数字或下划线，即[a-zA-Z0-9_]。可以使用标志r.A、r.ASCII，或者在表达式之前添加(?a)来指定ASCII模式（https://docs.python.org/3.3/library/re.html? highlight=re）。——译者注

注意，正则表达式要用到大量反斜杠（'\'），而反斜杠又是Python中的转义字符。因此，如果要将反斜杠作为常规字符处理，必须再加一个反斜杠（'\\'），这将导致正则表达式中出现大量的反斜杠，显得非常笨重。幸好Python支持原生字符串，在原生字符串中反斜杠不被解析为转义字符。

要定义原生字符串，只需要加一个r前缀。以下两个字符串是等价的，且两种表达式中都没有换行符：

```
"\\n"
r"\n"
```

本书将使用原生字符串的方式来书写正则表达式。

接下来我们学习一些有用的正则表达式。使用这些例子无意吓唬你，但通过它们你应该认识到，相比于生活的不易，计算机科学无疑是一门更加困难的学科，而其中最难的部分就是模式匹配。

- r"\w[-\w\.]*@\w[-\w]*(\.\w[-\w]*)+"

  这是一个电子邮件地址。

- r"<TAG\b[^>]*<(.*?)</TAG>"

  这是一个具有结束标签的HTML标签。

- r"[-+]?((\d*\.?\d+)|(\d\.))([eE][-+]?\d+)?"

  这是一个浮点数。

至此，你也许会一时冲动，打算写出一个正则表达式来匹配有效的网址，这个任务听上去确实很吸引人，但却是极其困难的。建议你抑制住冲动，直接使用urllib.parse模块，相关的内容已在第9单元中介绍过了。

不规则正则表达式



Python正则表达式不是唯一可用的正则表达式。Perl语言使用的正则表达式也具有同样的处理能力，但是语法和Python不同，语义上也略有差异。在一些简单情况下（例如文件名匹配），可以使用glob模块，它是另一种类型的正则表达式（参见第11单元）。

使用模块 re 进行搜索、拆分和替换

一旦你编写并编译了一个正则表达式，就可以用它来拆分、匹配、搜索和替换子字符串。re模块提供了所有必要的函数，且大多数函数接受两种类型的模式：原生的和编译后的。

```
re.function(rawPattern, ...)
compiledPattern.function(...)
```

函数split(pattern, string, maxsplit=0, flags=0)通过pattern将字符串拆分为至多maxsplit个子字符串，并返回子字符串列表（如果maxsplit==0，则返回所有子字符串）。如果你在寻找针对文本分析的分词器，那么这个函数可以作为一个免费的选择。

```
re.split(r"\W", "Hello, world!")
```

```
⇒ ['Hello', '', 'world', '']
```

```
# 合并所有相邻的非字母字符
re.split(r"\W+", "Hello, world!")
```

```
⇒ ['Hello', 'world', '']
```

函数match(pattern, string, flags=0)检查字符串的开头是否与正则表达式pattern相匹配。如果匹配成功，则该函数返回一个match对象，否则返回None。匹配对象（如果存在）可以使用start()、end()和group()函数，分别返回匹配片段的开始索引、结束索引以及片段本身。

```
mo = re.match(r"\d+", "067 Starts with a number")
```

```
⇒ <_sre.SRE_Match object; span=(0, 3), match='067'>
```

```
mo.group()
```

```
⇒ '067'
```

```
re.match(r"\d+", "Does not start with a number")
```

```
⇒ None
```

函数search(pattern, string, flags=0)检查整个字符串是否存在匹配正则表达式的部分。如果匹配成功，则该函数返回一个match对象，否则返回None。如果想要匹配的片段不在字符串的开头，就应使用search()函数替换上述的match()函数。

```
re.search(r"[a-z]+", "0010010 Has at least one 010 letter 0010010", re.I)
```

```
⇒ <_sre.SRE_Match object; span=(8, 11), match='Has'>
```

```
# 区分大小写的方式
re.search(r"[a-z]+", "0010010 Has at least one 010 letter 0010010")
```

```
⇒ <_sre.SRE_Match object; span=(9, 11), match='as'>
```

函数findall(pattern, string, flags=0)查找与正则表达式匹配的所有子字符串。该函数返回一个子字符串列表（当然，该列表可能为空）。

```
re.findall(r"[a-z]+", "0010010 Has at least one 010 letter 0010010", re.I)
```

```
⇒ ['Has', 'at', 'least', 'one', 'letter']
```

捕获组与非捕获组

非捕获组作为正则表达式的一部分，在re模块中被视为一种简单的记号①。非捕获组使用的括号与算术表达式中的括号在作用上并无差异。例如，r"cab+"匹配以"ca"开头、后跟至少一个"b"的子字符串，而r"c(?:ab)"匹配以"c"开头、后跟一个或多个"ab"的子字符串。注意，正则表达式中"(?:"和其他部分之间没有空格。

与非捕获组不同，捕获组除了具有分组的作用，还能捕获匹配的子字符串。例如，捕获组r"c(ab)+"在search()函数作用下，返回在"c"及其后至多个"ab"的子字符串，而在findall()函数作用下，返回一系列"ab"子字符串②。

函数sub(pattern, repl, string, flags=0)用repl替换字符串的所有非重叠匹配部分。使用可选参数count，可以限制替换的次数。

```
re.sub(r"[a-z ]+", "[...]", "0010010 has at least one 010 letter 0010010")
```

```
⇛ '0010010[...]010[...]0010010'
```

正则表达式非常强大，可是在许多情况下（例如，通过文件的后缀匹配文件名），使用正则表达式有点“杀鸡用牛刀”了。实际上，可以通过globbing实现类似的效果，具体内容请参考下一单元。

第 11 单元

globbing③文件名与其他字符串

globbing是匹配特定文件名和通配符的过程，是正则表达式的简化版。通配符可以包含特殊符号'*'（表示零个或多个字符）和'?'（表示正好一个字符）。请注意，'\'、'+'和'.'不是特殊符号!

glob模块提供了一个名称也为glob的、匹配通配符的函数。该函数返回与作为参数传递的通配符相匹配的所有文件名列表：

① 顾名思义，非捕获组是一种不具备“捕获”能力的组。换句话说，除了不能从组中获得匹配的内容之外，非捕获组跟正常的（捕获）组没有什么区别。——译者注

② 此处进一步给出一个例子：re.search(r"c(ab)+","cababjcabk")⇛'cabab' re.findall(r"c(ab)+","cababjcabk")⇛['ab','ab']。——译者注

③ globbing（通配）是黑客们的行话，通常表示把一个包含通配符的非具体文件名展开为（存储在计算机、服务器或者网络上的）具体文件名的过程。——译者注

```
glob.glob("*.txt")
```

⇒ **['public.policy.txt', 'big.data.txt']**

通配符'*'匹配当前目录（文件夹）中的所有文件名，以句点（'.'）开头的文件除外。如果要匹配这种以句点符号开头的特殊文件名，需使用通配符".*"。

第 12 单元

Pickling 和 Unpickling 数据

pickle模块用于实现**序列化**——将任意的Python数据结构保存到一个文件中，并将其作为Python表达式读回。可以使用任何Python程序读出文件中被pickle的表达式，但是用其他语言编写的程序做不到这一点（除非该语言实现了pickle协议）。

pickle文件必须以二进制读/写模式打开：

```
# 将一个对象转存（dump）到文件
with open("myData.pickle", "wb") as oFile:
  pickle.dump(object, oFile)

# 重新加载相同的对象
with open("myData.pickle", "rb") as iFile:
  object = pickle.load(iFile)
```

一个pickle文件中可以存储多个对象。load()函数能返回pickle文件中的下一个对象，而如果已到达文件结尾，则触发异常。也可以使用pickle来存储软件用不到的中间数据，这些软件通常不具备pickle的功能。

轮到你了

本章讲解了如何从本地磁盘文件和互联网中提取数据，用恰当的数据结构存储数据，提取与特定模式匹配的位和片段，以及pickle数据以便进一步处理。在计算机科学中，虽然知识和技术是有限的，但需要提取数据的场景却各式各样、无穷无尽。应用数据提取可以实现不同的目的，处理各种复杂的问题。此处仅列出有限的几种应用。

❑ **词频计数器***

编写一个程序，用于下载用户请求的网页，并给出网页中使用频率最高的十个词，所有词不区分大小写。出于练习的目的，可以简单地假设一个词由正则表达式r"\w+"确定[①]。

① 中文的分词比英文复杂得多，最好使用英文网页完成该练习。——译者注

- 文件索引器**

  编写一个程序，建立某个指定目录（文件夹）下所有文件的索引。程序应构造一个字典，其中键是所有文件中的所有唯一词（正则表达式`r"\w+"`所描述的、不区分大小写的词），并且字典里每个条目的值是包含该词的文件名列表。例如，如果单词aloha出现在文件early-internet.dat和hawaiian-travel.txt中，则字典将具有这样的条目：`{...,'aloha':['early-internet.dat','hawaiian-travel.txt'],...}`。

  另外，程序应对该字典执行pickle操作，以供将来使用。

- 电话号码提取器***

  编写一个程序，从给定的文本文件中提取出所有电话号码。这个任务并不容易，不同国家的电话号码书写格式超过几十种（请参考en.wikipedia.org/wiki/National_conventions_for_writing_telephone_numbers）。你能设计一个正则表达式来捕获它们吗？

  如果这个任务对你来说不是很难，那你可以试试提取地址！

这个杰森是谁？为什么诸神都喜欢他？他从哪里来？他有什么样的故事？

——希腊诗人荷马

第3章 使用文本数据

原始数据通常来自各种文本文档：结构化文档（HTML、XML、CSV和JSON文件）或非结构化文档（简单的、人类可读的文本）。事实上，非结构化文本可能是最难处理的数据源，因为处理软件必须推断出数据项的含义。

上一段中提到的所有数据表示都是人类可读的（这正是称它们为文本文档的原因）。必要时，可以用简单的文本编辑器（Windows上的Notepad、Linux上的gedit，以及Mac OS X上的TextEdit）打开任意的文本文件，进行阅读或者完成编辑。在没有其他可用工具的情况下，可以不去管具体的数据表示方法，而是将文本文档视为普通文本，直接使用核心Python的字符串函数来处理（参考第4单元）。

值得庆幸的是，Anaconda提供了几个优秀的模块——`BeautifulSoup`、`csv`、`json`和`nltk`——使原本枯燥的文本分析工作变得令人兴奋。按照**奥卡姆剃刀**原理——**如无必要，勿增实体**（这一原理实际上是由约翰·庞奇而不是由奥卡姆制定的），我们应该避免重新发明已经存在的工具。该原理不仅适用于文本处理工具，而且适用于所有Anaconda软件包。

本章通过简单的结构化数据开启文本数据处理的学习。然后，你将了解如何通过自然语言处理技术，向非结构化文本添加某些结构。

第13单元 处理HTML文件

本章介绍的第一种结构化文本文档是HTML——一种通常在Web上用于表示人类可读信息的标记语言。HTML文档包含文本以及用于控制文本的显示和释义的预定义标签（包含在尖括号<>

中）。标签可以具有属性，下表显示了一些HTML标签及其属性。

表3　一些常用的HTML标签和属性

标　签	属　性	用　途
HTML		整个HTML文档
HEAD		文档头
TITLE		文档标题
BODY	background、bgcolor	文档主体内容
H1、H2、H3等		小节头
I，EM		斜体强调样式
B，STRONG		加粗强调样式
PRE		格式化文本
P，SPAN，DIV		段落，块元素，行内元素
BR		换行
A	href	超链接
IMG	src，width，height	图片
TABLE	width，border	表格
TR		表格中的一行
TH，TD		表头/单元格
OL，UL		有序/无序列表
LI		列表
DL		描述列表
DT，DD		描述主题，描述定义
INPUT	name	用户输入域
SELECT	name	下拉菜单

HTML是XML的前身，人们发明它的初衷是使机器处理可读文档，它算不上是一门编程语言，只是一系列具有相似结构的标记语言。用户可以根据需要定义XML标签及其属性。

XML ≠ HTML



虽然XML和HTML看上去很相似，但一个典型的HTML文档通常不会是一个有效的XML文档。反过来，XML文档也不是HTML文档。

XML标签依赖于所使用的应用程序。只要遵循一些简单的规则（例如，包含在尖括号中），任何字母和数字组成的字符串都可以作为标签。XML标签只起到对文本进行解释描述的作用，而不控制文本的显示，因此常在不需要人类直接阅读的文档中使用。使用可扩展样式表语言转换（XSLT），能将XML转换为HTML，而使用级联样式表（CSS），可以给HTML文档添加样式。

BeautifulSoup模块可用于解析、访问以及修改HTML和XML文档。可以使用一个标记字符串、一个标记文件或一个标记文档的网址，来构建一个BeautifulSoup对象：

```
from bs4 import BeautifulSoup
from urllib.request import urlopen

# 使用字符串构建soup
soup1 = BeautifulSoup("<HTML><HEAD> «headers» </HEAD> «body» </HTML>")

# 使用本地文件构建soup
soup2 = BeautifulSoup(open("myDoc.html"))

# 使用Web文档构建soup
# 记住urlopen()不会添加"http://"!
soup3 = BeautifulSoup(urlopen("http://www.networksciencelab.com/"))
```

BeautifulSoup对象构造函数的第二个可选参数是标记解析器——负责提取HTML标签和实体的Python组件。BeautifulSoup附带四个预先安装好的解析器：

- "html.parser"（默认的解析器，解析速度非常快，但是规则较为严格，多用于“简单的”HTML文档）
- "lxml"（解析速度非常快，规则较宽松）
- "xml"（仅适用于XML文件）
- "html5lib"（解析速度非常慢，规则也非常宽松，用于处理具有复杂结构的HTML文档，而如果不考虑解析速度，可用于所有HTML文档）

soup准备就绪后，可以使用函数soup.prettify()完美地打印出原始标记文档。

函数soup.get_text()返回标记文档中去除了所有标签的文本部分。当我们感兴趣的内容是纯文本时，就可以使用这个函数实现标记文本向纯文本的转换。

```
htmlString = '''
  <HTML>
  <HEAD><TITLE>My document</TITLE></HEAD>
  <BODY>Main text.</BODY></HTML>
'''
soup = BeautifulSoup(htmlString)
soup.get_text()
```

```
⇒ 'nMy document\nMain text.n'
```

标签通常用于定位某些文件片段。例如，我们所感兴趣的部分可能是第一个表的第一行这样具体的片段。使用标签可以实现诸如此类的定位功能，尤其是当标签具有class或id属性时，而使用纯文本是不可能实现这个功能的。

BeautifulSoup对标签之间的垂直和水平关系使用统一的实现方法。类似于文件系统的层次结构，这种位置关系被表示为标签对象的属性。例如，soup的标题soup.title就是soup的对象属性。标题父元素的name对象的值是soup.title.parent.name.string，而第一个表第一行中

的第一个单元格大概能表示为`soup.body.table.tr.td`。

任何标签t都有一个名称`t.name`、一个字符串值（`t.string`表示原始内容，`t.stripped_strings`表示删除空白后的列表）、父标签`t.parent`、下一个标签`t.next`和前一个标签`t.prev`，以及零个或多个子标签`t.children`（标签中的标签）。

`BeautifulSoup`通过Python字典接口实现对HTML标签属性的访问。如果标签对象t表示超链接（例如`<a href="foobar.html">`），则超链接目标的字符串值为`t["href"].string`。HTML标签是不区分大小写的，这一点应引起注意。

soup最有用的函数应该就是`soup.find()`和`soup.find_all()`了，二者分别用于找到某个标签的第一个实例和所有实例。下面的例子说明了如何使用soup查询所需的内容。

- 标签`<H2>`的所有实例：

```
level2headers = soup.find_all("H2")
```

- 所有粗体或斜体格式：

```
formats = soup.find_all(["i", "b", "em", "strong"])
```

- 具有某个属性（例如`id="link3"`）的所有标签：

```
soup.find(id="link3")
```

- 使用字典符号或`tag.get()`函数，找出所有超链接以及第一个链接的目标网址：

```
links = soup.find_all("a")
firstLink = links[0]["href"]
# Or:
firstLink = links[0].get("href")
```

顺便提一下，如果标签不具备所查询的属性，则上面示例中的两种表达式都无法使用。鉴于可能遇到这样的情况，必须先使用`tag.has_attr()`函数检查属性是否存在，然后才能提取它。以下示例结合了`BeautifulSoup`和列表推导功能，来提取所有链接及其各自的网址和标签（这在用递归的方式抓取网页时非常有用）：

```
with urlopen("http://www.networksciencelab.com/") as doc:
  soup = BeautifulSoup(doc)

links = [(link.string, link["href"])
  for link in soup.find_all("a")
  if link.has_attr("href")]
```

链接的值是一个由元组对象构成的列表：

```
⇒ [('Network Science Workshop',
⇒  '<http://www.slideshare.net/DmitryZinoviev/workshop-20212296'),
⇒  «...»,('Academia.edu',
⇒  '<https://suffolk.academia.edu/DmitryZinoviev'),('ResearchGate',
⇒  '<https://www.researchgate.net/profile/Dmitry_Zinoviev')]
```

HTML/XML之所以强大，是因为它多样化的功能，但这种多功能性也未尝不是它的魔咒，尤其是涉及表格数据时。幸运的是，你可以在较为严谨但易于处理的CSV文件中存储表格数据，更多的内容请阅读下一个单元。

第 14 单元

处理 CSV 文件

CSV是一种结构化文本文件格式，用于存储和转移表格（或形式接近表格的）数据。该格式可以追溯到1972年，那时它是Microsoft Excel、Apache OpenOffice Calc和其他电子表格软件的首选格式。Data.gov[①]是一个提供公开数据的美国政府网站，该网站中CSV格式的数据集就有12 550个[②]。

一个CSV文件由表示变量的列和表示记录的行组成（具有统计背景的数据科学家通常称这些记录为观测值）。记录中的字段通常由逗号分隔，但其他分隔符也是比较常见的，例如制表符（制表符分隔值，TSV）、冒号、分号和竖直条等。建议在自己创建的文件中坚持使用逗号作为分隔符，同时保证编写的处理程序能正确处理使用其他分隔符的CSV文件。

值得注意的是，有时看起来像分隔符的字符并不是分隔符。通过将字段包含在引号字符中，可确保字段中的类分隔符字符作为变量值的一部分（如`...,"Hello, world",...`）。

为了使用方便，Python的`csv`模块提供了一个CSV读取器和一个CSV写入器。两个对象的第一个参数都是已打开的文本文件句柄（在下面的示例中，使用`newline=''`选项打开文件，从而避免删除行的操作）。必要时可以通过可选参数`delimiter`和`quotechar`，提供默认的分隔符和引号字符。Python还提供了控制转义字符、行终止符等定界符的可选参数。

```
with open("somefile.csv", newline='') as infile:
  reader = csv.reader(infile, delimiter=',', quotechar='"')
```

CSV文件的第一条记录通常包含列标题，可能与文件的其余部分有所不同。这只是一个常见的做法，并非CSV格式本身的特性。

CSV读取器提供了一个可以在`for`循环中使用的迭代器接口。迭代器将下一条记录作为一个字符串字段列表返回。读取器不会将字段转换为任何数值数据类型（必要时需要自己处理！），另外，除非传递可选参数`skipinitialspace=True`，否则不会删除前导的空白。

如果事先不知道CSV文件的大小，而且文件可能很大，则不宜一次性读取所有记录，而应使用增量的、迭代的、逐行的处理方式：读出一行，处理一行，再获取另一行。

① catalog.data.gov/dataset?res_format=CSV
② 翻译此书时该值为14 234。——译者注

CSV写入器提供`writerow()`和`writerows()`两个函数。`writerow()`将一个字符串或数字序列作为一条记录写入文件。该函数将数字转换成字符串，因此不必担心数值表示的问题。类似地，`writerows()`将字符串或数字序列的列表作为记录集写入文件。

在下面的示例中，使用`csv`模块从CSV文件中提取Answer.Age列。假设此列肯定存在，但列的索引未知。一旦获得数值，借助`statistics`模块就能得到年龄的平均值和标准偏差。

首先，打开文件并读取数据：

```
with open("demographics.csv", newline='') as infile:
  data = list(csv.reader(infile))
```


检查文件中的第一个记录 `data[0]`，它必须包含感兴趣的列标题：

```
ageIndex = data[0].index("Answer.Age")
```

最后，访问剩余记录中感兴趣的字段，并计算和显示统计数据：

```
ages = [int(row[ageIndex]) for row in data[1:]]
print(statistics.mean(ages), statistics.stdev(ages))
```

`csv`和`statistics`模块是底层的、快速而粗糙的工具。在第6章，你将了解如何在更为复杂的项目中使用`pandas`的数据frame，完成那些比对几列数据进行琐碎的检索要高端得多的任务。

第 15 单元

读取 JSON 文件

JSON是一种轻量级的数据交换格式。该格式跟编程语言无关，这一点不同于`pickle`模块（第12单元提过），但在数据表示方面，JSON有更多的限制，不如`pickle`灵活。

什么人在使用JSON？



许多流行的网站，比如Twitter[①]、Facebook[②]和Yahoo! Weather[③]，都提供使用JSON作为数据交换格式的API。

JSON支持以下数据类型。

- 原子数据类型——字符串、数字、`true`、`false`和`null`
- 数组——使用方括号`[]`表示，对应Python的列表，数组各项可以使用不同的数据类型：

① dev.twitter.com/overview/documentation

② developers.facebook.com

③ developer.yahoo.com/weather/

```
[1, 3.14, "a string", true, null]
```

- 对象——使用花括号（{}）表示，对应Python的字典，对象各项由冒号分隔的键和值组成：

```
{"age" : 37, "gender" : "male", "married" : true}
```

- 数组、对象和原子数据类型的任何递归组合（对象数组、项为数组的对象，等等）

有一个不尽如人意之处，那就是某些Python数据类型和结构（比如集合和复数）无法存储在JSON文件中。因此，要在导出到JSON之前，将它们转换为JSON可表示的数据类型。例如，将复数存储为两个double类型的数字组成的数组，将集合存储为一个由集合的各项所组成的数组。

将复杂数据存储到JSON文件中的操作称为**序列化**，相应的反向操作则称为**反序列化**。Python通过`json`模块中的函数，实现JSON序列化和反序列化。

函数`dump()`将一个能用JSON表示的Python对象导出（dump）到先前打开的文本文件。函数`dumps()`导出的Python对象为文本字符串（这是为了打印出美观的结果或实现进程间通信）。`dump()`和`dumps()`这两个函数都实现了序列化。

pickling的意义



当数据保存到JSON文件时，只是保存了变量的值，重新加载后，这些值都是相互独立的。当使用pickle来保存同样的数据时，保存的是原始变量的引用，重新加载后，对同一变量的所有引用关系维持不变。

函数`loads()`将有效的JSON字符串转换为Python对象（即将对象"加载"到Python中）。实际中，总会遇到这样的转换。类似地，函数`load()`将已打开的文本文件的内容转换为一个Python对象。把多个对象存储在一个JSON文件中是一种错误的做法，但如果已有的文件包含多个对象，则可将其以文本的方式读入，进而将文本转换为对象数组（在文本中各个对象之间添加方括号和逗号分隔符），并使用`loads()`将文本反序列化为对象列表。

以下代码片段实现了将任意（可序列化的）对象按先序列化、后反序列化的顺序进行处理：

```
object = «some serializable object»
# 将对象保存到文件
with open("data.json", "w") as out_json:
  json.dump(object, out_json, indent=None, sort_keys=False)
# 从文件载入对象
with open("data.json") as in_json:
  object1 = json.load(in_json)
# 将对象序列化为字符串
json_string = json.dumps(object1)
# 把字符串解析为JSON
object2 = json.loads(json_string)
```

尽管经历了四次"痛苦"的转换，`object`、`object1`和`object2`仍然具有相同的值。

当预期到结果可能会被进一步处理或导入另一个程序时，通常都使用JSON格式来存储最终处理结果。

第 16 单元 处理自然语言中的文本

根据经验，所有潜在的可用数据中，约80%的数据是非结构化的，其中包括音频、视频、图像（这些内容超出了本书的范围）和用自然语言编写的文本[1]。自然语言中的文本没有标签、分隔符和数据类型，但它仍然是丰富的信息源。有时我们想知道某些词是否出现在文本中及出现的频率（**词句标记**），文本属于什么类别（**文本分类**），它传达了正面的还是负面的消息（**情感分析**），文本中提到的人和物（**实体提取**），等等。如果只是一两个文本，我们尚且可以靠肉眼来读取和处理，但对于大规模的文本分析，就必须借助于自然语言处理（NLP）。

3

Python的`nltk`模块（自然语言工具包）中实现了很多NLP的功能。该模块围绕语料库（字词和表达式的集合）、函数和算法进行组织。

NLTK 语料库

语料库是单词或表达式的结构化或非结构化集合。所有NLTK语料库都存储在模块`nltk.corpus`中。下面是一些例子。

- `gutenberg`——包含来自古登堡计划的十八个英语文本，例如《白鲸》和《圣经》。
- `names`——8000名男性和女性名字的列表。
- `words`——235 000个最常用的英语单词和表单的列表。
- `stopwords`——以十四种语言列出的停用词（即最常用的词）列表。英文列表位于`stopwords.words("english")`中。大多数的文本分析都要删除停用词，因为通常它们对于文本的理解不会带来新的信息。
- `cmudict`——诞生在卡内基梅隆大学的发音字典（也因此而得名），有134 000多个条目。`cmudict.entries()`中的每个条目都是一个单词和一个音节列表组成的元组，同一个词可能有几种不同的发音。这个语料库可用于识别同音词（soundalikes）。

`nltk.corpus.wordnet`对象是另一个语料库的接口——一个在线语义词网络WordNet（需要访问互联网）。该网络是用词性和序列号标记的**同义词**集合：

```
wn = nltk.corpus.wordnet # 语料库读取器
wn.synsets("cat")
⇒ [Synset('cat.n.01'), Synset('guy.n.01'), «more synsets»]
```

可以查找每个同义词的定义，这可能是一个意想不到的功能：

[1] www.informationweek.com/software/information-management/structure-models-and-meaning/d/d-id/1030187

```
wn.synset("cat.n.01").definition()
wn.synset("cat.n.02").definition()
```

```
⇒ 'feline mammal usually having thick soft fur «...»'
⇒ 'an informal term for a youth or man'
```

同义词可以具有上义词（含义较为抽象的同义词）和下义词（含义较为具体的同义词），这些功能使得同义词看起来像具有子类和超类的面向对象（OOP）类。

```
wn.synset("cat.n.01").hypernyms()
wn.synset("cat.n.01").hyponyms()
```

```
⇒  [Synset('feline.n.01')]
⇒  [Synset('domestic_cat.n.01'), Synset('wildcat.n.03')]
```

最后，可以使用WordNet来计算两个同义词组之间的语义相似性。相似度是范围[0...1]中的双精度数。如果相似性为0，则同义词是无关的；如果相似性为1，则同义词是完全同义词。

```
x = wn.synset("cat.n.01")
y = wn.synset("lynx.n.01")
x.path_similarity(y)
```

```
⇒ 0.04
```

上面提到的是两个词之间的相似性，那么任意两个词之间的相似性如何分析呢？让我们来看看“dog”和“cat”的所有同义词，并找到语义最接近的定义：

```
[simxy.definition() for simxy in max(
  (x.path_similarity(y), x, y)
    for x in wn.synsets('cat')
    for y in wn.synsets('dog')
    if x.path_similarity(y) # 确保synsets是相关的
  )[1:]]
```

```
⇒ ['an informal term for a youth or man', 'informal term for a man']
```

真是太神奇了！

除了使用标准语料库，还可以通过`PlaintextCorpusReader`创建自己的语料库。`PlaintextCorpusReader`会在根目录中查找与`glob`模式匹配的文件。

```
myCorpus = nltk.corpus.PlaintextCorpusReader(root, glob)
```

函数`fileids()`返回包含在新创建的语料库中的文件列表。函数`raw()`返回语料库中原始的“原始”文本。函数`sents()`返回所有句子的列表。函数`words()`返回所有单词的列表。在下一节中，你将领略到把原始文本转换为句子和单词的神奇魔法。

```
myCorpus.fileids()
myCorpus.raw()
myCorpus.sents()
myCorpus.words()
```

使用下一小节介绍的函数并结合Counter对象（参考第7单元），可以计算单词出现的频率并识别最常用的单词。

NLTK是一个空模块

安装模块NLTK时，实际上只安装了类，而未安装语料库。该语料库太大，以至于被认为无法包含在发行版中。第一次导入模块时，需要调用download()函数（需要连接互联网），并根据需求安装缺失的部件。

规范化

规范化是用自然语言的表达形式对文本进行整理，以便进一步处理的过程。规范化通常包括以下步骤（不一定严格参照这些步骤，但基本上是按此顺序处理的）。

(1) 分词（将文本切分成单词）。NLTK提供了两个简单的和两个较为高级的分词器。句子切分器返回一个列表，列表元素为句子字符串。其他所有分词器返回的都是一个单词列表：

- word_tokenize(text)——切分单词
- sent_tokenize(text)——切分句子
- regexp_tokenize(text, re)——基于正则表达式的分词，参数re是描述有效字的正则表达式

根据分词器的质量和句子的标点结构，有些“词”可以包含非字母字符。对于严重依赖于标点符号的分析任务（例如通过表情符号的情感分析），需要用到更为高级的WordPunctcTokenizer。下面是WordPunctTokenizer.tokenize()和word_tokenize()对相同文本解析结果的对比：

```
from nltk.tokenize import WordPunctTokenizer
word_punct = WordPunctTokenizer()
text = "}Help! :))) :[ ..... :D{"
word_punct.tokenize(text)
```

```
⇒ ["}", "Help", "!", ":)))", ":[", ".....", ":", "D", "{"]
```

```
nltk.word_tokenize(text)
```

```
⇒ ["}", "Help", "!", ":", ")", ")", ")", ":", "[", "...",
⇒   "..", ":", "D", "{"]
```

(2) 将单词转换为大小写相同的字符（全部大写或小写）。

(3) 删除停用词。使用语料库中的停用词和其他特定的停用词列表作为参考。需要注意的是，停用词使用的是小写字母。语料库中并不存在“THE”这个停用词（虽然它肯定是停用词）。

(4) 词干化处理（将各种词形转换为词干）[①]。NLTK提供两个基本的词干分析器：较为保守的Porter和较为激进的Lancaster。由于其激进的规则，Lancaster会产生更多的音形相近但含义不同的词干。两个分析器都有`stem(word)`，这个函数会返回所谓的词干：

```
pstemmer = nltk.PorterStemmer()
pstemmer.stem("wonderful")
```

⇒ **'wonder'**

```
lstemmer = nltk.LancasterStemmer()
lstemmer.stem("wonderful")
```

⇒ **'wond'**

两个词干分析器的特点在于：仅将词干分析应用于单个单词，而不是完整短语。

(5) 词形还原——一种速度更慢也更保守的词干分析机制。`WordNetLemmatizer`查找WordNet中算出的词干，且仅接受单词或表格形式的词干。（要使用lemmatizer，需要访问互联网。）函数`lemmatize(word)`返回`word`表示的词的词元。

```
lemmatizer = nltk.WordNetLemmatizer()
lemmatizer.lemmatize("wonderful")
```

⇒ **'wonderful'**

另外，虽然在技术层面上，词性标注（POS标注）并不属于规范化的范畴，但它也是文本预处理中的重要步骤。函数`nltk.pos_tag(text)`为文本中的每个单词分配一个词性标签，其中文本是一个单词列表。函数的返回值是一个元组列表，其中元组的第一个元素是原始单词，第二个元素是标签。

```
nltk.pos_tag(["beautiful", "world"])
# 一个形容词和一个名词
```

⇒ **[('beautiful', 'JJ'), ('world', 'NN')]**

下面的代码从文件index.html中找出了10个最常用的非停用词的词干，这个例子涵盖了上面提到的知识点，以及在第13单元介绍的`BeautifulSoup`。

```
from bs4 import BeautifulSoup
from collections import Counter
from nltk.corpus import stopwords
from nltk import LancasterStemmer

# 创建一个新词干
ls = nltk.LancasterStemmer()

# 读取文件并生成soup
```

① 对于英文而言，所谓的词干化处理主要包括把名词的复数形式变成单数，将其他时态变成基本形态等类似的处理。——译者注

```
with open("index.html") as infile:
  soup = BeautifulSoup(infile)

# 提取并标记文本
words = nltk.word_tokenize(soup.text)

# 转换为小写
words = [w.lower() for w in words]

# 删除停止单词，并分析剩余部分的词干
words = [ls.stem(w) for w in text if w not in
  stopwords.words("english") and w.isalnum()]

# 对词进行计数
freqs = Counter(words)
print(freqs.most_common(10))
```


本段代码可以作为实现主题抽取的第一步。

其他文本处理程序

关于其他高级NLP程序的讨论超出了本书范围，此处简要地罗列几项。

- 分段——在没有特定的单词边界语法的文本中（例如，中文文本中）对“单词”的边界进行识别。可以将分段方法应用于任何字符或数字序列（例如，购物清单、DNA片段等）。
- 文本分类——根据预设的条件将文本划分到某一已知类别中。情绪分析是文本分类的一种特殊情况，通常以情绪词的出现频率作为分析的基础。
- 实体提取——检测具有预定属性的单词或短语，通常指实体，例如人、地理位置、公司、产品和品牌。
- 潜在语义索引——使用奇异值分解（SVD）在非结构化文本集合中识别出术语与概念之间关系的模式。顺便指出，SVD在统计学上又被称为主成分分析（PCA）。

轮到你了

至此，你已知道如何从现有的HTML、XML、CSV或JSON文件中（甚至从纯文本中）提取有价值的数据，也了解了HTML和XML标签及其结构，能从数据中分离标签，并规范化字词（至少可以实现一定程度的规范化）。懂得这些后，你可以完成很多大项目了——当然你还需要一些耐心。现在让我们来做一些练习！

- **失效链接检测器***

  编写一个程序，基于一个给定网页的URL，报出页面中失效链接的名称和地址。出于练习的目的，约定当使用`urllib.request.urlopen()`打开链接失败时，则认为链接失效。

❑ **维基百科矿工****

MediaWiki（一个维基百科项目[1]）提供了基于JSON的API，可以实现对维基百科数据和元数据的可编程访问。编写一个程序，报出标题为“Data science”的维基百科页面中使用频率最高的十个词干。

实现提示如下。

- 使用HTTP，而不是HTTPS。
- 阅读MediaWiki网站上的“simple example”，并将其作为程序的基础。
- 首先，按标题获取页面ID，然后按ID获取页面。
- 阅读JSON数据，留意不同层次的密钥：在写作本书时，答案具有六个层次的深度！

❑ **音乐流派分类器*****

编写一个程序，使用维基百科计算不同的摇滚/流行音乐流派之间的语义相似性。从按类型[2]分类的主要音乐组的列表开始。（注意，列表是分层的，而且包含子类！）递归处理列表及其子列表，直到找出所有相关组（为了节省时间和流量，可以通过选取固定的子类别，例如英国摇滚类，实现对搜索的限制）。如果可能的话，对于每个发现的组，提取其类型。使用Jaccard相似性指数[3]作为语义相似性的度量：对于每一个类型组A和B，$J(A,B)=|A\cap B|/|A\cup B|=|C|/(|A|+|B|-|C|)$，其中|A|、|B|分别是列出类型A、B的组的数量，|C|则是同时列出A和B的组的数量。将运行结果保存（pickle），以备将来使用。相信我，你不会愿意多次运行这个程序的！

顺便提一下，本程序计算出共有多少种类型以及最相关的类型是什么？

① www.mediawiki.org/wiki/API:Main_page
② en.wikipedia.org/wiki/Category:Rock_music_groups_by_genre
③ en.wikipedia.org/wiki/Jaccard_index

哪些是女神？……第十个是Vor，她是如此聪明，没有什么可以瞒得了她。

——冰岛历史学家、诗人和政治家Snorri Sturluson

第 4 章

使用数据库

4

你学习（或实践）了数据科学，也掌握了如何将磁盘文件中的原始数据以Python数据结构导入。让我们趁热打铁，开启数据科学的另一个主题——**数据库**——一个用于长期存储数据的工具。

数据库是整个数据分析过程中的重要组成部分。

- 通常，输入数据是以数据库表的形式提供的。为实现进一步的处理，必须从数据库中将数据检索出来。
- 数据库能实现高度优化、快速且非易失性的数据存储，可用于存储原始数据、中间结果和最终结果（无论原始数据是否存储在数据库中）。
- 数据库提供高度优化的数据转换，包括排序、选择和连接功能。如果数据库中已经存在原始数据或中间结果，则还可以实现数据聚合。

本章将结合MySQL和MongoDB（或NoSQL数据库）——当前最流行的关系数据库之一和最流行的文档存储——探索如何设置、配置、填充和查询数据。你很可能已经知道怎么将这些数据库作为预配置和预填充的数据源了，但进一步了解数据库引擎丰富的内部世界依然是有益的。它不仅能使你成为一名更优秀的程序员，也能为学习后续章节中的pandas模块（第6章）奠定坚实的基础。

第 17 单元

设置 MySQL 数据库

关系数据库是用于实现数据永久存储的一组表，同时还可能具有数据排序和索引的功能。关系数据库非常适合存储表格数据（例如CSV文件中的数据），其中每一个表代表一种变量类型，

表的列对应变量，表的行对应观测值或记录。

虽然不使用Python也可以操作数据库，但你至少需要知道结构化查询语言（SQL）或其特定实现（例如MySQL），这是通过命令行或Python应用程序访问关系数据库的基础。

另外，你需要安装一个MySQL客户端（例如mysql），才能通过命令行与运行中的MySQL数据库服务器进行交互。所有MySQL命令都是不区分大小写的，而且每条命令必须以分号结尾。

以数据库管理员的身份使用mysql启动一个新的数据库项目（这些操作只需要执行一次）。

(1) 在shell命令行中启动mysql：

```
c:\myProject> mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
«More mysql output»
mysql>
```

后续所有的指令都在mysql命令行提示符下输入。

(2) 创建新的数据库用户（“dsuser”）和密码（“badpassw0rd”）：

```
CREATE USER 'dsuser'@'localhost' IDENTIFIED BY 'badpassw0rd';
```

(3) 给项目创建一个新数据库（“dsdb”）：

```
CREATE DATABASE dsdb;
```

(4) 授予新用户对新数据库的访问权限：

```
GRANT ALL ON dsdb.* TO 'dsuser'@'localhost';
```

现在，可以在数据库dsdb中创建新表了。使用同样的mysql客户端，以常规用户身份登录：

```
c:\myProject> mysql -u dsuser -p dsdb
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
«More mysql output»
mysql>
```

通常，表都是一次创建而多次访问的。创建表之后，可以根据项目需求更改其属性。CREATE TABLE命令创建一个新表，后跟表名和所有的列信息组成的列表。每个列信息按列名称和该列数据类型的顺序进行定义。最常见的MySQL数据类型是TINYINT、SMALLINT、INT、FLOAT、DOUBLE、CHAR、VARCHAR、TINYTEXT、TEXT、DATE、TIME、DATETIME和TIMESTAMP。

以下命令使用empname（可变长度的文本）、salary（浮点数）和hired（日期）列创建表employee。表中的每个记录代表一个员工。

```
USE dsdb;
CREATE TABLE employee (empname TINYTEXT, salary FLOAT, hired DATE);
```

```
⇒ Query OK, 0 rows affected (0.17 sec)
```

对于不再需要的表，可将其从数据库中删去。

```
DROP TABLE employee;
```

```
⇛ Query OK, 0 rows affected (0.05 sec)
```

DROP命令简短精炼，且不可撤销。正所谓覆水难收，在删除任何东西之前，请务必考虑清楚！

数据库架构



数据库架构是描述所有表、列、数据类型、索引、约束以及不同表之间关系的数据库结构。架构是数据库的核心：将一个数据库的所有表中的数据删除后，剩下的就是数据库的架构。

只要存储空间允许，应使得每条记录都能自动生成主键和自动更新最后修改的时间戳，尽管这并非任何编程语言的标准。主键可以确保记录的唯一性，还可以加快搜索的速度。最后修改的时间戳使数据具备一定的可追溯性，而关键字NOT NULL标记的列确保每条记录在该列都具备一个有效值：

```
CREATE TABLE employee (id INT PRIMARY KEY AUTO_INCREMENT,
  updated TIMESTAMP, empname TINYTEXT NOT NULL, salary FLOAT NOT NULL,
  hired DATE);
```

如果想使用列（变量）进行排序、搜索或连接，就需要给该列添加索引：

```
ALTER TABLE employee ADD INDEX(hired);
```

```
⇛ Query OK, 0 rows affected (0.22 sec)
⇛ Records: 0 Duplicates: 0 Warnings: 0
```

注意，索引大大增加了查询的时间，同时也显著增加了插入和删除的时间。另外，只有在将大量数据插入表中之后，才能创建索引。如果要插入新的数据，首先需要删除已有的索引：

```
DROP INDEX hired ON employee;
```

然后才能插入数据并重新添加索引。

如果某一列中所有的值都是唯一的（例如员工ID号或姓名），那就应该给该列添加UNIQUE约束。如果UNIQUE约束的列使用了长度可变的数据类型（例如VARCHAR、TINYTEXT或TEXT），则必须指定相应的长度：

```
ALTER TABLE employee ADD UNIQUE(empname(255));
```

主键总是有一个值（使用NOT NULL标记），而且它既是一个INDEX，也是UNIQUE的。

第 18 单元

使用 MySQL 数据库：命令行

MySQL支持五种基本的数据库操作：插入、删除、变更、选择和连接。这些操作可以完成数据库表的填充，以及修改和检索现有数据。这些操作通常包含在数据分析程序中，但为了建立对它们的认识，我们将首先在mysql命令行提示符下练习它们。

插入

插入是最重要的命令，我们就从它开始介绍。下面我们将插入一条新的记录到一张表中，并重复这一操作，直到所有的观测值都被记录到表中：

```
INSERT INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

```
⇒ Query OK, 1 row affected, 1 warning (0.18 sec)
```

前两个NULL是索引和时间戳的占位符值，服务器会自动算出它们的值。函数NOW()返回当前日期和时间，但是只有“日期”部分会写入记录中。注意，查询产生了一个警告，该警告的来源就是对后者（“时间”）的截断。下面的代码用于查看最近的警告和错误的文字描述：

```
SHOW WARNINGS;
```

```
⇒ +-------+------+---------------------------------------------+
⇒ | Level | Code | Message                                     |
⇒ +-------+------+---------------------------------------------+
⇒ | Note  | 1265 | Data truncated for column 'hired' at row 1 |
⇒ +-------+------+---------------------------------------------+
⇒ 1 row in set (0.00 sec)
```

如果插入操作违反了UNIQUE约束，服务器就会中止操作，除非使用了IGNORE关键字。在这种情况下，插入操作失败：

```
INSERT INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

```
⇒ ERROR 1062 (23000): Duplicate entry 'John Smith' for key 'empname'
```

```
INSERT IGNORE INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

```
⇒ Query OK, 0 rows affected, 1 warning (0.14 sec)
```

可以手动插入更多的行，不过首选方法是用Python来完成其余的插入。

删除

删除操作从表中删除所有与搜索条件匹配的记录。如果没有指定搜索条件，服务器就会删除所有记录：

```
-- 如果John Smith是一个低收入者，就将他的记录删除
DELETE FROM employee WHERE salary<11000 AND empname="John Smith";
-- 删除所有记录
DELETE FROM employee;
```

如果只想删除某条特定的记录，则应使用其唯一主键或任何其他的具有唯一性标识的条件：

```
DELETE FROM employee WHERE id=387513;
```

需要强调的是，任何时候都要记住，删除操作是不可撤销的！

变更

变更操作会找到与搜索条件相匹配的记录，更新其指定列的值。如果未指定搜索条件，则变更操作将影响所有记录：

```
-- 重置最近招入的员工工资
UPDATE employee SET salary=35000 WHERE hired=CURDATE();
-- 再次增加John Smith的工资
UPDATE employee SET salary=salary+1000 WHERE empname="John Smith";

⇛ Query OK, 1 row affected (0.06 sec)
⇛ Rows matched: 1 Changed: 1 Warnings: 0
```

想必你已经猜到了：变更也是不可撤销的。实际上，它跟删除操作一样，是一个破坏性的操作。

选择

选择操作在与搜索条件匹配的所有记录中选出所有请求的列。如果未指定搜索条件，则将获得所有记录；这种操作得到的数据记录的数量可能远超出你的需求。

```
SELECT empname,salary FROM employee WHERE empname="John Smith";

⇛ +------------+--------+
⇛ | empname    | salary |
⇛ +------------+--------+
⇛ | John Smith | 36000  |
⇛ +------------+--------+
⇛ 1 row in set (0.00 sec)

SELECT empname,salary FROM employee;

⇛ +--------------+--------+
⇛ | empname      | salary |
⇛ +--------------+--------+
```

```
⇒ | John Smith   | 36000  |
⇒ | Jane Doe     | 75000  |
⇒ | Abe Lincoln  | 0.01   |
⇒ | Anon I. Muss | 14000  |
⇒ +-------------+--------+
⇒ 4 rows in set (0.00 sec)
```

可以通过对结果进行排序、分组、聚合和过滤来增强选择的功能。若想对选择结果排序，需要使用ORDER BY修饰符（可实现多列数据的降序排列或升序排列）：

```
SELECT * FROM employee WHERE hired>='2000-01-01' ORDER BY salary DESC;
```

```
⇒ +----+---------------------+-------------+--------+------------+
⇒ | id | updated             | empname     | salary | hired      |
⇒ +----+---------------------+-------------+--------+------------+
⇒ | 4  | 2016-01-09 17:35:11 | Jane Doe     | 75000  | 2011-11-11 |
⇒ | 1  | 2016-01-09 17:31:29 | John Smith   | 36000  | 2016-01-09 |
⇒ | 6  | 2016-01-09 17:55:24 | Anon I. Muss | 14000  | 2011-01-01 |
⇒ +----+---------------------+-------------+--------+------------+
⇒ 3 rows in set (0.01 sec)
```

若想对选择结果进行分组和聚合，需要使用GROUP BY修饰符和聚合函数（比如COUNT()、MIN()、MAX()、SUM()或AVG()）：

```
SELECT (hired>'2001-01-01') AS Recent,AVG(salary)
  FROM employee
  GROUP BY (hired>'2001-01-01');
```

```
⇒ +--------+---------------------+
⇒ | Recent | AVG(salary)         |
⇒ +--------+---------------------+
⇒ |      0 | 0.009999999776482582 |
⇒ |      1 |   41666.666666666664 |
⇒ +--------+---------------------+
⇒ 2 rows in set (0.00 sec)
```

后一个语句计算并给出每组员工的平均工资，分组的依据是员工的招聘时间在2001年1月1日之前还是之后，也就是招聘时间本身。

使用WHERE和HAVING关键字可以对选择结果进行过滤。服务器在分组之前执行WHERE命令，并在分组之后执行HAVING命令。

```
SELECT AVG(salary),MIN(hired),MAX(hired) FROM employee
  GROUP BY YEAR(hired)
  HAVING MIN(hired)>'2001-01-01';
```

```
⇒ +-------------+------------+------------+
⇒ | AVG(salary) | MIN(hired) | MAX(hired) |
⇒ +-------------+------------+------------+
⇒ |       44500 | 2011-01-01 | 2011-11-11 |
⇒ |       36000 | 2016-01-09 | 2016-01-09 |
⇒ +-------------+------------+------------+
⇒ 2 rows in set (0.00 sec)
```

这条语句计算并给出2001年1月1日后同一年度内招聘员工的平均工资，同时给出每个年度最早的和最新的员工招聘时间。

连接

连接操作基于一列或多列，完成两个表的合并。MySQL支持五种类型的连接：内连接（具有所谓的直接连接的含义）、左连接、右连接、外连接和自然连接。后者也可以是左连接或右连接。当两个表中存在至少一个匹配时，内连接返回匹配的行。左连接/右连接分别连接左/右表中的所有行，即使在另一侧没有匹配，也能实现连接。外连接返回在任意表中具有匹配的行。如果一个表没有匹配，则服务器返回NULL。自然连接的行为类似于外连接，不同之处在于它隐式涉及名称相同的所有列。

以下命令创建包含员工职位的新表，然后为准备用于连接的列添加索引，最后从两个表中提取员工姓名和职位（后面的操作使用了隐式内连接的语法）：

4

```
-- 准备并填充另一张表
CREATE TABLE position (eid INT, description TEXT);
INSERT INTO position (eid,description) VALUES (6,'Imposter'),
  (1,'Accountant'),(4,'Programmer'),(5,'President');
ALTER TABLE position ADD INDEX(eid);

-- 获取连接后的数据
SELECT employee.empname,position.description
  FROM employee,position WHERE employee.id=position.eid
  ORDER BY position.description;

⇛ +--------------+-------------+
⇛ | empname      | description |
⇛ +--------------+-------------+
⇛ | John Smith   | Accountant  |
⇛ | Anon I. Muss | Imposter    |
⇛ | Abe Lincoln  | President   |
⇛ | Jane Doe     | Programmer  |
⇛ +--------------+-------------+
⇛ 4 rows in set (0.00 sec)
```

第 19 单元

使用 MySQL 数据库：pymysql

Python使用数据库驱动模块与MySQL通信。诸如pymysql等许多数据库驱动都是免费的。本单元将使用pymysql，它是Anaconda的一部分。驱动程序经过激活后与数据库服务器相连，然后将Python的函数调用转换为数据库查询，反过来，将数据库结果转换为Python数据结构。

connect()函数需要以下信息：数据库（名称）、数据库服务器的位置（主机和端口号）和数据库用户（名称和密码）。如果数据库成功连接，则返回连接标识符。接下来，创建与数据库连接相关联的数据库游标：

```
conn = pymysql.connect(host="localhost", port=3306,
  user="dsuser", passwd="badpassw0rd", db="dsdb")
cur = conn.cursor()
```

游标的execute()函数向数据服务器提交要执行的查询命令，并返回受影响的行数（如果查询是非破坏性的，则返回零）。查询命令只是一个字符串，其构建方法参考上一单元。与命令行MySQL查询不同，pymysql查询语句不需要在结尾加上分号。

```
query = '''
SELECT employee.empname,position.description
  FROM employee,position WHERE employee.id=position.eid
  ORDER BY position.description
'''
n_rows = cur.execute(query)
```

如果提交非破坏性查询（比如SELECT），需要使用游标函数fetchall()获取所有匹配的记录。该函数返回一个生成器，可以将其转换为列字段的元组构成的列表：

```
results = list(cur.fetchall())
```

```
⇒ [('John Smith', 'Accountant'), ('Anon I. Muss', 'Imposter'),
⇒  ('Abe Lincoln', 'President'), ('Jane Doe', 'Programmer')]
```

如果查询是破坏性的（例如UPDATE、DELETE或INSERT），则必须执行commit操作。（注意，提供commit()函数的是连接本身，而不是游标。）

```
conn.commit()
```

如果在一个破坏性查询之后没有执行commit，服务器就不会修改表。

关系数据库自1974年以来沿用至今（Ingres）①。它们在数据规范化方面展开了很多伟大的工作，使得数据可以自然地分解为表、列和行，这可以说是关系数据库带给人们的馈赠。其实，任何数据集都可以被规范化，但是规范化的代价可能非常大（无论是在实现规范化方面，还是在最终查询性能方面）。某些类型的数据是不遵循规范化约定的，比如文本文档、图像、音频和视频剪辑，以及不规则数据结构。对于这些数据，不应强制它们遵循SQL规范，而应选择一个NoSQL文档来存储它们。接下来，我们就来介绍相关的内容。

① uickbase.intuit.com/articles/timeline-of-database-history

第 20 单元

改善文档存储：MongoDB

文档存储（即一个NoSQL数据库）是一个具有属性的对象（通常称为文档）的非易失性集合。目前，人们已经开发了许多不同的文档存储的实现方案。本单元重点分析其中一种实现方案——MongoDB，并简要介绍其最主要的竞争对手CouchDB[1]。

MongoDB是一个非关系型数据库。一个MongoDB服务器可以支持多个不相关的数据库。数据库由一个或多个文档集合组成，集合中的所有文档都有唯一的标识符。

在Python中，我们用pymongo模块中MongoClient类的实例来实现MongoDB客户端。可以创建一个无参数的客户端（适用于典型的安装了本地服务器的情况），也可以用服务器的主机名和端口号作为参数创建客户端，或使用服务器的统一资源标识符（URI）作为参数创建客户端：


```
import pymongo as mongo
# 使用默认的初始化方式
client1 = mongo.MongoClient()
# 指定主机和端口号
client2 = mongo.MongoClient("localhost", 27017)
# 用URI方式指定主机和端口号
client3 = mongo.MongoClient("mongodb://localhost:27017/")
```

客户一旦端建立了与数据库服务器的连接，就可以选择当前激活的数据库，进而选择激活的集合。您可以使用面向对象（“点”）或字典样式的符号。如果所选的数据库或集合不存在，服务器会立即创建它们：

```
# 创建并选择活动数据库的两种方法
db = client1.dsdb
db = client1["dsdb"]

# 创建并选择活动集合的两种方法
people = db.people
people = db["people"]
```

pymongo模块用字典变量来表示MongoDB文档。表示对象的每个字典必须具有_id这个键。如果该键不存在，服务器会自动生成它。

集合对象提供用于在文档集合中插入、搜索、删除、更新、替换和聚合文档以及创建索引的功能。

函数insert_one(doc)和insert_many(docs)将文档或文档列表插入集合。它们分别返回对象InsertOneResult或InsertManyResult，这两个对象分别提供inserted_id和inserted_ids

① ouchdb.apache.org

属性。当文档没有明确的键时，就需要使用这些属性来找出文档的键。如果指定了_id键，则在执行插入操作后，该键值也不会改变：

```
person1 = {"empname" : "John Smith", "dob" : "1957-12-24"}
person2 = {"_id" : "XVT162", "empname" : "Jane Doe", "dob" : "1964-05-16"}
person_id1 = people.insert_one(person1).inserted_id

⇒ ObjectId('5691a8720f759d05092d311b')

# 注意出现了一个新的名为“_id”的键！
person1

⇒ {'empname': 'John Smith', 'dob': '1957-12-24',
⇒   '_id': ObjectId('5691a8720f759d05092d311b')}

person_id2 = people.insert_one(person2).inserted_id

⇒ "XVT162"

persons = [{"empname" : "Abe Lincoln", "dob" : "1809-02-12"},
           {"empname" : "Anon I. Muss"}]
result = people.insert_many(persons)
result.inserted_ids

⇒ [ObjectId('5691a9900f759d05092d311c'),
⇒  ObjectId('5691a9900f759d05092d311d')]
```

函数find_one()和find()给出匹配可选属性的一个或多个文档，其中find_one()返回文档，而find()返回一个游标（一个生成器），可以使用list()函数将该游标转换为列表，或者在for循环中将其用作迭代器。如果将字典作为参数传递给这些函数中的任意一个，函数将给出与字典的所有键值相等的文档：

```
everyone = people.find()
list(everyone)

⇒ [{'empname': 'John Smith', 'dob': '1957-12-24',
⇒   '_id': ObjectId('5691a8720f759d05092d311b')},
⇒   {'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'},
⇒   {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒   '_id': ObjectId('5691a9900f759d05092d311c')},
⇒   {'empname': 'Anon I. Muss', '_id': ObjectId('5691a9900f759d05092d311d')}]

list(people.find({"dob" : "1957-12-24"}))

⇒ [{'empname': 'John Smith', 'dob': '1957-12-24',
⇒   '_id': ObjectId('5691a8720f759d05092d311b')}]

people.find_one()

⇒ [{'empname': 'John Smith', 'dob': '1957-12-24',
⇒   '_id': ObjectId('5691a8720f759d05092d311b')}]
```

```
people.find_one({"empname" : "Abe Lincoln"})

⇒ {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒  '_id': ObjectId('5691a9900f759d05092d311c')}

people.find_one({"_id" : "XVT162"})

⇒ {'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'}
```

下面介绍几个实现数据聚合和排序的分组和排序函数。函数sort()对查询的结果进行排序。当以无参数的方式调用它时，该函数按键_id的升序进行排序。函数count()返回查询结果中或整个集合中的文档数量：

```
people.count()

⇒ 4

people.find({"dob": "1957-12-24"}).count()

⇒ 1

people.find().sort("dob")

⇒ [{'empname': 'Anon I. Muss', '_id': ObjectId('5691a9900f759d05092d311d')},
⇒  {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒   '_id': ObjectId('5691a9900f759d05092d311c')},
⇒  {'empname': 'John Smith', 'dob': '1957-12-24',
⇒   '_id': ObjectId('5691a8720f759d05092d311b')},
⇒  {'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'}]
```

函数delete_one(doc)和delete_many(docs)从集合中删除字典doc所标识的一个或多个文档。如果要在删除所有文档的同时保留集合，需使用空字典作为参数调用函数delete_many({})：

```
result = people.delete_many({"dob" : "1957-12-24"})
result.deleted_count

⇒ 1
```

CouchDB

CouchDB是另外一个流行的NoSQL数据库。与MongoDB不同，CouchDB更侧重于可用性而非一致性。对于复制的CouchDB数据库（在多台计算机上运行），所有用户都可以使用它，但却不能保证不同的用户得到的文档是相同的。而对于复制的MongoDB，用户得到的一定是完全相同的文档，但是某些用户可能会无法使用数据库。如果实际使用中不复制数据库，那究竟是采用CouchDB还是MongoDB，就完全取决于个人喜好了。

轮到你了

数据库管理是一个重要的科学领域，它的博大精深远远超出了本书的范围。仅阅读本章无法使你成为一名经验丰富的数据库管理员或全能的数据库程序员。不过至少你现在已经能创建一两个表，将数据存储到表中，并在必要时恢复数据，而且你还知道了两种实现方法：使用SQL和不使用SQL。

- MySQL文件索引器*

  编写一个Python程序，对于给定文件中的每个单词，记录如下信息到MySQL数据库中：单词本身（不是词干！）、单词在文件中的序数（从1开始），以及单词的词性标记。使用NLTK `WordPunct-Tokenizer`（参考第16单元第2小节）来识别单词。假设这些单词比较短，数据类型可以使用MySQL的`TINYTEXT`。设计数据库模式，创建所有必需的表，并在正式开始编写Python代码之前，通过命令行来试用一下设计出来的表。

- MySQL到MongoDB的转换器**

  MySQL语句`DESCRIBE table_name`能给出表中所有列的名称、数据类型、约束、默认值等。编写一个Python程序，将所有数据从用户指定的一个MySQL表中转移到一个MongoDB文档。要求程序不能修改时间戳。

还没有获得数据就进行推理，这是个致命的错误。

——英国作家Arthur Conan Doyle爵士

第5章 使用表格形式的数值数据

各种文本文档是原始数据的主要来源，而这些文本的格式通常以数值为主。例如Excel和CSV电子表格，尤其是数据库表，它们可能包含数百万或数十亿的数值记录。Python核心无疑是一款优秀的文本处理工具，但有时候它的数值运算性能却不尽如人意，numpy模块就是为了弥补这方面的缺陷而诞生的。

NumPy（Numeric Python，以numpy导入）是一系列高效的、可并行的、执行高性能数值运算的函数的接口。numpy模块提供了一种新的Python数据结构——数组（array），以及特定于该结构的函数工具箱。该模块还支持随机数、数据聚合、线性代数和傅里叶变换等非常实用的数值计算工具。

仙境之桥



如果程序需要访问大量的数值数据（例如TB量级，甚至更大），那就一定会用到h5py模块[①]。该模块是HDF5二进制数据格式的入口，可以与许多第三方软件（比如IDL和MATLAB）配合使用。h5py模仿了大家所熟悉的numpy和Python的一些机制，例如数组和字典。只要学会了numpy，就能很自然地运用h5py，因此本书不再介绍h5py的相关内容。

在本章中，你将学习如何创建不同形状的numpy数组，基于不同的源创建numpy数组，数组的重排和切片操作，添加数组索引，以及对某些或所有数组元素进行算术运算、逻辑运算和聚合运算。

① www.h5py.org

第 21 单元

创建数组

numpy数组比原生的Python列表更为紧凑和高效，尤其是在多维的情况下。但与列表不同的是，数组的语法要求更为严格：数组必须是同构的。这意味着数组项不能混合使用不同的数据类型，而且不能对不同数据类型的数组项进行匹配操作。

创建numpy数组的方法很多。可以使用函数array()，基于类数组（array-like）数据创建数组。所谓的类数组数据可以是列表、元组或另一个数组。numpy基于数据本身推断出数组元素的类型，当然，你也可以给array()传递确定的dtype参数。numpy支持的数据类型接近二十种，例如bool_、int64、uint64、float64和<U32（针对Unicode字符串）。

为获得较高的效率，numpy在创建一个数组时，不会将数据从源复制到新数组，而是建立起数据间的连接。也就是说，在默认情况下，numpy数组相当于是其底层数据的视图，而不是其副本。如果底层数据对象发生改变，则相应的数组数据也会随之改变。如果你不喜欢这种方式（这是默认的处理方式，除非复制的数据量过大），可以给构造函数传递copy=True 。

列表即数组，数组亦是数组



实际上，Python的“列表”（list）是以数组的方式实现的，而并非列表的方式[1]，这与“列表”（list）的字面含义并不一致。由于未使用前向指针，所以Python并没有给列表预留前向指针的存储空间。Python的大型列表只比“真正的”numpy数组多使用约13%的存储空间，但对于一些简单的内置操作，比如sum()，使用列表则要比数组快五到十倍。因此在使用numpy之前，应该问问自己是否真的需要用到某些numpy特有的功能！

接下来，我们来创建第一个数组——前10个正整数组成的简单数组：

```
import numpy as np
numbers = np.array(range(1, 11), copy=True)
```

```
⇒ array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

函数ones()、zeros()和empty()分别构造全1数组、全零数组和尚未初始化的数组。这些函数必须有数组的形状参数，该参数用一个与数组的维度相同的列表或元组来表征。

```
ones = np.ones([2, 4], dtype=np.float64)
```

```
⇒ array([[ 1., 1., 1., 1.],
```

① 这里的列表实现方式可以理解为链表（linked list）。——译者注

```
⇒          [ 1., 1., 1., 1.]])

zeros = np.zeros([2, 4], dtype=np.float64)

⇒ array([[ 0., 0., 0., 0.],
⇒        [ 0., 0., 0., 0.]])

empty = np.empty([2, 4], dtype=np.float64)
# 用这种方式创建的数组，其元素值不一定为零!

⇒ array([[ 0., 0., 0., 0.],
⇒        [ 0., 0., 0., 0.]])
```

numpy使用数组的ndim、shape和dtype属性分别存储数组的维数、形状和数据类型。

```
ones.shape # 只要没有经过变形（reshape），该属性给出的就是数组的原始形状

⇒ (2, 4)

numbers.ndim # 等价于len(numbers.shape)

⇒ 1

zeros.dtype

⇒ dtype('float64')
```

函数eye(N, M=None, k=0, dtype=np.float)用于构造一个N×M的**眼形**单位矩阵，其第*k*对角线上的值为1，其他地方的值为零。当k为正数时，对应的对角线位于主对角线上方的第*k*条。M为None（默认值）等价于M=N。

```
eye = np.eye(3, k=1)

⇒ array([[ 0., 1., 0.],
⇒        [ 0., 0., 1.],
⇒        [ 0., 0., 0.]])
```

当需要将几个矩阵相乘时，可以使用单位矩阵作为乘法链累积器中的初始值。

除了经典的内置函数range()外，numpy有其独有的、更高效的生成等间隔数值数组的方式：函数arange([start,] stop [, step,], dtype=None)。

```
np_numbers = np.arange(2, 5, 0.25)

⇒ array([ 2. , 2.25, 2.5 , 2.75, 3. , 3.25, 3.5 , 3.75, 4. ,
⇒         4.25, 4.5 , 4.75])
```

和range()函数一样，stop的值可以小于start，但必须保证step为负数且数组中的数值按顺序减小。

numpy会在创建数组时记录每一项的数据类型，不过该数据类型并非不可变的。可在数组创建后，调用函数astype(dtype, casting ="unsafe", copy=True)来改变它。对于类型缩小

的情况（将较抽象的数据类型转换为更具体的数据类型），可能会丢失一些信息。这并非numpy特有的，任何缩小变换都可能会丢失信息。

```
np_inumbers = np_numbers.astype(np.int)

⇒ array([2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4])
```

大多数numpy操作（例如第22单元讨论的转置操作）返回的是一个视图，而非原始数组的副本。为了保留原始数据，可使用copy()函数创建现有数组的副本。这样一来，对原始数组的任何更改都不会影响到副本。但如果数组较为庞大，比如有十亿个数组项，那就不要轻易进行复制。

```
np_inumbers_copy = np_inumbers.copy()
```

现在，让我们向更加高级的操作迈进。

第 22 单元

转置和重排

numpy数组的发明具有里程碑式的意义。借助numpy可以很容易地改变数组的形状和方向，我们再也不用像“瞎猫碰到死耗子”那样看运气了。下面我们用几个标准普尔（S&P）股票代码组成一个一维数组，然后用所有可能的方式改变它的形状：

```
# 几个标准普尔（S&P）股票代码
sap = np.array(["MMM", "ABT", "ABBV", "ACN", "ACE", "ATVI", "ADBE", "ADT"])

⇒ array('MMM', 'ABT', 'ABBV', 'ACN', 'ACE', 'ATVI', 'ADBE', 'ADT'],
⇒       dtype='<U4')
```

函数reshape(d0, d1, ...)可以改变现有数组的形状，其参数定义了新的维度。形状参数的维度必须维持不变，这是一条在numpy中普遍成立的法则。

```
sap2d = sap.reshape(2, 4)

⇒ array([['MMM', 'ABT', 'ABBV', 'ACN'],
⇒        ['ACE', 'ATVI', 'ADBE', 'ADT']],
⇒       dtype='<U4')

sap3d = sap.reshape(2, 2, 2)

⇒ array([[['MMM', 'ABT'],
⇒          ['ABBV', 'ACN']],
⇒
⇒         [['ACE', 'ATVI'],
⇒          ['ADBE', 'ADT']]],
⇒       dtype='<U4')
```

你甚至不需要调用函数，就能实现数组的转置：属性T的值就是数组的转置视图（对于一维数组，data.T==data；对于二维数组，转置运算将交换矩阵的行和列）。

```
sap2d.T

⇒ array([['MMM', 'ACE'],
⇒        ['ABT', 'ATVI'],
⇒        ['ABBV', 'ADBE'],
⇒        ['ACN', 'ADT']],
⇒       dtype='<U4')
```

本质上，属性T是通过重新标记十字基准线来显示矩阵的，具体方法如下：轴数为0的（“垂直”）变轴数为1（“水平”），反之亦然。函数swapaxes()是属性T的一个更通用的版本。它以任意两个轴作为参数，通过交换二者来实现多维数组的转置。自然地，传递二维数组的轴0和轴1也可以转置数组，就像使用属性T一样，这是数组转置的一种简单情况。

```
sap3d.swapaxes(1, 2)

⇒ array([[['MMM', 'ABBV'],
⇒         ['ABT', 'ACN']],
⇒
⇒        [['ACE', 'ADBE'],
⇒         ['ATVI', 'ADT']]],
⇒       dtype='<U4')
```

numpy还有一个实现转置的函数，即transpose()，它的通用性甚至比swapaxes()还高（尽管该函数名的含义与属性T相似，但二者实际上是不同的方法）。transpose()根据多维数组的参数（该参数必须是元组）来排列多维数组的部分或所有轴。在下面的示例中，使用transpose()保持第一个轴“垂直”，同时交换另外两个轴。

```
sap3d.transpose((0, 2, 1))

⇒ array([[['MMM', 'ABBV'],
⇒         ['ABT', 'ACN']],
⇒
⇒        [['ACE', 'ADBE'],
⇒         ['ATVI', 'ADT']]],
⇒       dtype='<U4')
```

这个结果恰好与swapaxes(1,2)的结果相同！

5

第 23 单元 索引和切片

numpy数组支持同Python列表一致的索引[i]和切片[i:j]操作。此外，数组还能实现布尔索

引：可以使用布尔值数组作为索引，其结果是原始数组中布尔索引为True的项构成的数组。通常，布尔数组是广播的结果。布尔索引既可以在右侧（用于选择），也可以在左侧（用于分配）。

假设我们从数据提供者处得知，脏数据集中的所有数据都是非负数。这就意味着任何负数都不是真实值，而是错误值，必须将其替换为更有意义的数（比如零），这样的操作被称为**数据清洗**。由此可见，要清除脏数据，首先需要找出有问题的值，然后运用合理的替代方式替换它们。

```
dirty = np.array([9, 4, 1, -0.01, -0.02, -0.001])
whos_dirty = dirty < 0 # 这是一个布尔数组，可以作为布尔索引来使用
```

```
⇒ array([False, False, False, True, True, True], dtype=bool)
```

```
dirty[whos_dirty] = 0 # 将所有的负值设置为0
```

```
⇒ array([9, 4, 1, 0, 0, 0])
```

可以将多个布尔表达式与以下运算符组合使用：|（逻辑或）、&（逻辑与）以及-（逻辑非）。如果想知道以下列表中哪些项目介于–0.5和0.5之间，直接“问”numpy吧！

```
linear = np.arange(-1, 1.1, 0.2)
(linear <= 0.5) & (linear >= -0.5)
```

```
⇒ array([False, False, False, True, True, True, True, True, False,
⇒        False, False], dtype=bool)
```

两种布尔运算



关系运算符（例如<和==）的优先级低于位运算符&、|和!，它们是作用在numpy数组上的“布尔”运算符。这常常令人感到困惑，因为“正常的”Python布尔运算符（or 、and和not）的优先级是比关系运算符还低的。为了确保numpy数组的布尔运算能首先运行，必须将其包含在括号中。

numpy数组的另一个很酷的特性是“智能”索引和“智能”切片，其中索引不是一个标量，而是一个数组或索引列表。其结果是索引对应的项组成的数组。下面我们从之前建立的S&P列表中选出第二个、第三个和最后一个股票代码（这就是所谓的“智能”索引）。

```
sap[[1, 2, -1]]
```

```
⇒ array(['ABT', 'BBV', 'ADT'],
⇒       dtype='<U4')
```

有时候你可能会想：为什么不从重构的数组中提取中间列的所有行呢？（这就是所谓的“智能”切片。）有如下两种实现方法：

```
sap2d[:, [1]]
```

```
⇒ array([['ABT'],
⇒        ['ATVI']],
⇒       dtype='<U4')
```

```
sap2d[:, 1]
```

```
⇒ array(['ABT', 'ATVI'],
⇒       dtype='<U4')
```

Python针对一个问题会提供多种相似的工具，这是极好的，但是一定要小心不要选错工具。仔细对比前面挑选出的两个数组可以发现，第一个是二维矩阵，而第二个是一维数组。结合你的需求，肯定只能选择其中之一。因此，在刚开始使用Python时，有必要检查一下结果，确保它真的就是你想要的。

第 24 单元

广播

numpy对矢量化算术运算的支持非常友好，只需保证参与运算的数组具有相同的形状。如果想在不使用numpy的前提下实现两个数组元素相加，就必须使用for循环或列表推导式，而如果使用numpy，则只需直接将它们相加即可：

```
a = np.arange(4)
b = np.arange(1, 5)
a + b
```

```
⇒ array([1, 3, 5, 7])
```

数组上的矢量化操作又被称为广播。如果参与运算的两个数组维度相等（如上所述）或其中一个是标量（如下所示），就可以将运算在两个维度“广播”：

```
a * 5
```

```
⇒ array([0, 5, 10, 15])
```

累计还是相乘？



在Python和numpy中，星号运算符（*）具有不同的行为。在“核心”Python中，表达式seq*5将列表seq复制五次，而同样的numpy表达式则将数组seq的每个元素乘以5。

你可以把数组和标量混合使用，并配上不同的算术运算。下面让我们创建一个对角矩阵，并添加一些小的（但不是随机的）噪声：

```
noise = np.eye(4) + 0.01 * np.ones((4, ))
```

```
⇒ array([[ 1.01, 0.01, 0.01, 0.01],
⇒        [ 0.01, 1.01, 0.01, 0.01],
```

```
⇒        [ 0.01, 0.01, 1.01, 0.01],
⇒        [ 0.01, 0.01, 0.01, 1.01]])
```

如何得到一些小的随机噪声呢？我们将在第47单元详细讨论随机数生成器，这里只给出一个例子：

```
noise = np.eye(4) + 0.01 * np.random.random([4, 4])
np.round(noise, 2)
```

```
⇒  array([[ 1.01, 0.  , 0.01, 0.  ],
⇒         [ 0.01, 1.01, 0.  , 0.01],
⇒         [ 0.  , 0.  , 1.  , 0.  ],
⇒         [ 0.  , 0.  , 0.01, 1.  ]])
```

使用**通用函数**round()对矩阵元素执行四舍五入操作——一个函数作用在所有的数组元素上！第25单元“揭秘通用函数”可以帮助你真正掌握通用函数（ufunc）。

顺便提一下，如果多次运行前面的示例，会得到不同的结果。这是因为随机数是随机的！

我们将在第30单元“生成合成正弦波”中详细讨论真实世界的、噪声的，以及正弦波形式的信号生成，并使用高级的图形化方式进行研究。

第 25 单元

揭秘通用函数

矢量化通用函数是对应广播功能的函数。只需要一次函数调用，就可以将通用函数应用于所有数组项。numpy提供了很多通用函数，举例如下。

- 算术运算：add()、multiply()、negative()、exp()、log()、sqrt()
- 三角函数：sin()、cos()、hypot()
- 按位运算：bitwise_and()、left_shift()
- 关系和逻辑运算：less()、logical_not()、equal()
- maximum()和minimum()
- 使用浮点数的函数：isinf()、isfinite()、floor()、isnan()

假设一个名为stocks的一维数组中记录了sap中8个代码对应的股票在2016年1月10日的周末前后的价格。

```
Stocks
```

```
⇒ array([ 140.49,    0.97,   40.68,   41.53,   55.7 ,   57.21,   98.2 ,
⇒          99.19,  109.96,  111.47,   35.71,   36.27,   87.85,   89.11,
⇒          30.22,   30.91])
```

现在我们想知道哪些股票的价格在周末下跌了。首先，按照股票代码对股价进行分组，并将较新的报价放在较早的报价之后。其实现方法是把原始数组重排成2×8的矩阵：

```
stocks = stocks.reshape(8, 2).T

⇒ array([[ 140.49,    40.68,   55.7 ,   98.2 ,   109.96,   35.71,   87.85,
⇒           30.22],
⇒        [   0.97,    41.53,   57.21,   99.19,   111.47,   36.27,   89.11,
⇒           30.91]])
```

接下来使用`greater()`函数逐列比较数组的两行数据，根据所得到的布尔索引就能找出感兴趣的股票代码：

```
fall = np.greater(stocks[0], stocks[1])

⇒ array([True, False, False, False, False, False, False, False], dtype=bool)

sap[fall]

⇒ array(['MMM'],
⇒       dtype='<U4')
```

顺便介绍一下，MMM是一家使用“庞氏骗局”的俄罗斯公司，它从未在任何证券交易所上市。难怪它的股价正在下跌。

除了“传统”数字之外，`numpy`全面支持IEEE 754浮点标准，并提供正无穷大（`inf`）和非数字（`nan`）两种符号。如果没有使用`numpy`，则这两个符号分别是`float("inf")`和`float("nan")`。按照数据科学领域约定的做法，本书使用nan作为缺失数据的占位符（缺失数据的问题在第1单元中介绍过）。

通用函数`isnan()`是一个极好用的nan的定位工具。实际上，对于下面的例子，用零值替换缺失的数据恐怕是一个很糟糕的做法（我们之前在第23单元就是这么做的），所以下面我们用`isnan()`重做一次：

```
# 假定MMM股票的新报价丢失了
stocks[1, 0] = np.nan
np.isnan(stocks)

⇒ array([[False, False, False, False, False, False, False, False],
⇒        [ True, False, False, False, False, False, False, False]], dtype=bool)

# 修复；没有比这更糟的做法了
stocks[np.isnan(stocks)] = 0

⇒ array([[ 140.49,    40.68,   55.7 ,   98.2 ,   109.96,   35.71,   87.85,
⇒           30.22],
⇒        [   0.  ,    41.53,   57.21,   99.19,   111.47,   36.27,   89.11,
⇒           30.91]])
```

通用函数扩展了Python的算术函数和关系运算符的功能，增加了更多的可能性，这种方式给出的条件函数可以说是Python逻辑运算符的加强版。

第 26 单元

理解条件函数

函数where(c, a, b)是numpy风格的if-else三元运算符[1]。它通过一个布尔数组（c）和两个其他数组（a和b）得出数组d，数组d满足：如果c[i]为真，则d[i]=a[i]，否则d[i]=b[i]。三个数组必须具有相同的形状。

如果数组中任意元素为真，则函数any()返回True。如果数组中所有元素为真，则函数all()返回True。

函数nonzero()返回所有非零元素的索引。

我们在第25单元用数组sap中记录了一些标准普尔股票的价格。现在要找出哪支股票的变化幅度较大（每股超过1.00美元），我们将幅度变化“小”的用零替换，然后找出非零元素，将它们的索引作为“智能索引”作用在股票符号的数组上：

```
changes = np.where(np.abs(stocks[1] - stocks[0]) > 1.00,
                   stocks[1] - stocks[0], 0)
```

```
⇒ array([-139.52,    0. ,     1.51,    0. ,     1.51,    0. ,    1.26,    0.  ])
```

```
sap[np.nonzero(changes)]
```

```
⇒ array(['MMM',  'ABBV',  'ACE',  'ADBE'],
⇒       dtype='<U4')
```

单独使用布尔索引也可以得到相同的结果：

```
sap[np.abs(stocks[1] - stocks[0]) > 1.00]
```

```
⇒ array(['MMM', 'ABBV', 'ACE', 'ADBE'],
⇒       dtype='<U4')
```

但是这样做就少了很多乐趣!

第 27 单元

数组的聚合与排序

数据排序和聚合在数据科学中处于核心地位。数据科学研究通常从大量的数据开始，逐渐通

[1] if-else条件表达式是从Python 2.5开始引入的写法。X if C else Y：只有当条件C为真时，X才会被执行；只有当C为假时，Y才会被执行。具体的讨论见Python官网：https://www.python.org/dev/peps/pep-0308/。——译者注

过组合、平均、积累等方式来提炼数据，直到数据有望被分解为一个小型的、易于呈现和解释的集合。numpy模块提供了计算numpy数组的各种聚集量的函数：mean()、sum()、std()（标准方差）、min()和max()。

下面我们使用广播、聚合函数、通用函数和布尔索引的组合（几乎是我们的整套工具集），从第25单元给出的股票中，提取出股价变化超过所有8支股票平均值的投资组合：

```
sap[         np.abs(stocks[0] - stocks[1])
    > np.mean(np.abs(stocks[0] - stocks[1]))]

⇒ array(['MMM'],
⇒       dtype='<U4')
```

说实话，其实不应该把股价变化大于零和小于零的混在一起考虑。

函数cumsum(x)和cumprod(x)分别计算累积和以及乘积量：$cumsum_i=\Sigma_1^i x_i$和$cumprod_i=\Pi_1^i x_i$。可以将它们分别用作数据的加法（简单利息支付）和乘法（复利支付）积分器。（注意，如果数组的任意元素为0，则该元素对应的cumprod()元素及其后续所有元素都等于0。）

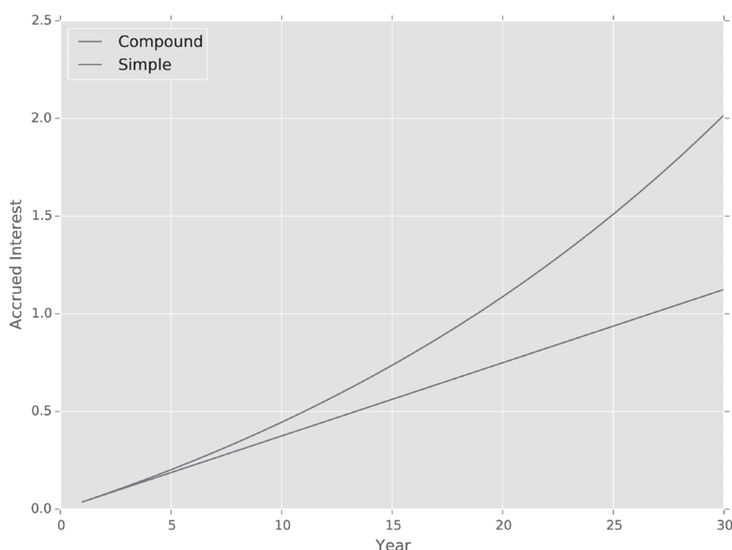
下面给出利率为3.75%时，近30年来的简单利息和复利的对比——只需要两行numpy代码加上绘制费用和税费：

5

邮电

interest.py

```
# 仅列出部分对比结果
RATE = .0375
TERM = 30
simple = (       RATE  * np.ones(TERM)).cumsum()
compound = ((1 + RATE) * np.ones(TERM)).cumprod() - 1
```



sort()函数可能是numpy模块中最没意思的函数。它只是将数组进行排序（覆盖原始数组顺序），并返回None。如果原始数组对你来说是比较宝贵的，那么在排序之前要先做一个副本。

第 28 单元 将数组用作集合

在某些情况下，数组内各项的顺序并不重要，重要的是这些项的组合，例如判断数组中是否存在某个特定的项，或者数组中存在什么类型的项。使用numpy可以将数组当作数学中的集合那样进行处理。

函数unique(x)返回x中所有唯一元素组成的数组。它是Counter模块的一个很好的替代品（关于Counter模块的内容，请参考第7单元“使用计数器”），但它并未真正计算元素出现的次数。

众所周知，数据科学几乎使生物信息学成了最好的学科（这真是自切片面包被发明以来最好的东西啦！[1]）。在生物信息学领域，人们使用数据科学来处理基因组测序，也就是确定DNA核苷酸的顺序。下面让我们做一些类似于在生物信息学中做的工作，找出在随机DNA片段中存在的核苷酸种类：

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna_as_array = np.array(list(dna))
```

```
⇒ array(['A', 'G', 'T', 'C', 'C', 'G', 'C', 'G', 'A', 'A', 'T', 'A', 'C',
⇒        'A', 'G', 'G', 'C', 'T', 'C', 'G', 'G', 'T'],
⇒       dtype='<U1')
```

```
np.unique(dna_as_array)
```

```
⇒ array(['A', 'C', 'G', 'T'],
⇒       dtype='<U1')
```

想必你已经明白了[2]。

函数in1d(needle, haystack)返回一个布尔数组，如果needle的元素在haystack中，则返回数组的对应位置元素为True。数组needle和haystack不需要具有相同的形状。

```
np.in1d(["MSFT", "MMM", "AAPL"], sap)
```

```
⇒ array([False, True, False], dtype=bool)
```

函数union1d()和intersect1d()计算两个一维数组的并集和交集，数组的大小可以不相

① 切片面包意义重大，以至于英语中出现了这样的表达：“The best thing since sliced bread.”——译者注

② www.genomenewsnetwork.org/resources/whats_a_genome/Chp2_1.shtml

同。不过，相信你会更愿意使用Python原生的集合运算符&和|，因为二者的运行速度大约是numpy的两倍！

第 29 单元 数组的保存和读取

随着学习的深入，你以后可能不会单独使用numpy，而是把它作为pandas（第6章）、networkx（第7章）和机器学习工具（第10章）的强大后端。通常的做法是，基于低级的数据处理工具提供的数据创建numpy数组，再将它们提交给更高级的分析工具。因此，你未必需要直接保存或读取numpy数组。

如果非要单独使用numpy也是可以的，numpy提供了内置的函数，能够将数组保存到.npy文件（函数save(file,arr)），并从.npy文件中读取以前保存的数组（函数load(file)）。.npy文件是二进制格式的，只有numpy可以处理它们。

也许你已经意识到，这是两个没什么用处的函数，不过它们还是非常友好的：传入的file参数可以是打开的文件句柄或字符串形式的文件名。如果文件名不存在.npy扩展也没关系，函数能自动添加扩展。

```
# 一种愚蠢的数组复制方法
np.save("sap.npy", sap)
sap_copy = np.load("sap")
```

另一对函数loadtxt()和savetxt()能从文本文件加载表格数据，并将数组保存到文本文件。numpy会自动创建文件，而且必要时可以将其自动打开。如果文件名以.gz结尾，numpy甚至可以自动压缩和解压文件。你还可以设定numpy处理注释行与分隔符的方式，并跳过不需要的行：

```
arr = np.loadtxt(fname, comments="#", delimiter=None, skiprows=0,
                 dtype=float)
np.savetxt(fname, arr, comments="#", delimiter=" ", dtype=float)
```

第 30 单元 生成合成正弦波

现在我们来做一些数据科学家通常不会做的事情，打动那些不使用numpy的朋友：生成一个**合成正弦波**，我们可能已经在廉价的、不完美的，以及嘈杂的仪器中收到过小量的这种周期信号。这样的信号具体是从何处获得的以及仪器的性质并不重要。产生这类信号的仪器也许是一个插在

电源插座上的电压表，或者是一个在周日晚放在草坪上而直到周五才收回的户外数字温度计，甚至可能是一个股票市场价格报价器。（顺便说一句，合成周期信号不仅可用于给朋友留下印象，还可用于测试新的数字信号处理算法。）

生成信号的代码如下，其中高亮显示的部分就是使用numpy的代码。这部分代码见证了矢量化的魔法：创建一个连续整数数组，将它们转换为浮点数，调整为正确的周期，取正弦，放大，置换，添加高斯噪声（参见第45单元），并模拟仪器测量得到的信号截断效果。

numpy_sinewave.py

```
# 导入所有优质的模块
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

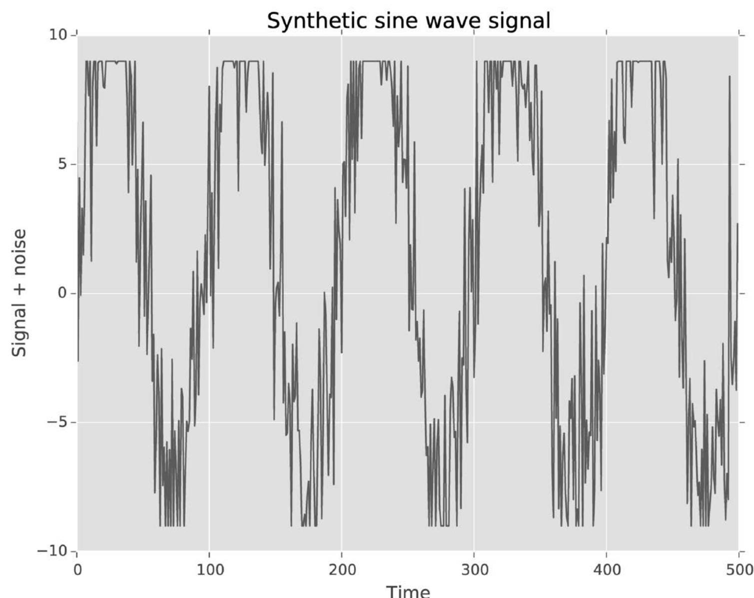
# 定义了信号、噪声和“仪器”属性的常量
SIG_AMPLITUDE = 10; SIG_OFFSET = 2; SIG_PERIOD = 100
NOISE_AMPLITUDE = 3
N_SAMPLES = 5 * SIG_PERIOD
INSTRUMENT_RANGE = 9

➤ # 创建一个正弦波，并加入随机噪声
➤ times = np.arange(N_SAMPLES).astype(float)
➤ signal = SIG_AMPLITUDE * np.sin(2 * np.pi * times / SIG_PERIOD) + SIG_OFFSET
➤ noise = NOISE_AMPLITUDE * np.random.normal(size=N_SAMPLES)
➤ signal += noise
➤
➤ # 将仪器测量范围之外的峰值截断
➤ signal[signal > INSTRUMENT_RANGE] = INSTRUMENT_RANGE
➤ signal[signal < -INSTRUMENT_RANGE] = -INSTRUMENT_RANGE

# 绘制结果
matplotlib.style.use("ggplot")
plt.plot(times, signal)
plt.title("Synthetic sine wave signal")
plt.xlabel("Time")
plt.ylabel("Signal + noise")
plt.ylim(ymin = -SIG_AMPLITUDE, ymax = SIG_AMPLITUDE)

# 保存图片
plt.savefig("../images/signal.pdf")
```

其他未高亮显示的代码使用Matplotlib实现噪声信号的可视化。



Matplotlib作为numpy的姐妹包，是第8章“绘图”的重点内容。

轮到你了

希望本章的学习，已使你确信numpy是一个处理数字的优秀工具箱。矢量和矩阵是numpy最重要的成员。numpy提供矢量化的算术运算、逻辑运算和其他操作，还提供用于数据重排、排序和聚合的功能。它甚至支持用nan来表示一个非数的值。你现在能使用numpy驾驭一些计算密集型的项目了吗？

- **数组微分器***

  （函数的）部分和近似等价于（函数的）积分。事实上，微积分理论就把积分定义为无穷小元素的无限求和。（函数的）部分差arr_{i+1}–arr_i近似等价于（函数的）导数。numpy没有提供用于计算数组部分差的工具。请编写一个程序，对于一个给定的数组arr，计算数组项的部分差。在本练习中假设数组为数值数组。

- **HEI定位器****

  从www.data.gov/education下载美国高等教育数据集的CSV文件。编写一个程序，给出最接近整个数据集给出的平均经纬度的十个高等教育机构（HEI）的地理位置。计算距离时，以度为单位。尽量使用numpy进行数据存储和处理。请注意，CSV文件的第一行包含列标题，而且文件中的某些字段甚至整个记录可能是无效的。

- 状态相似性计算器**

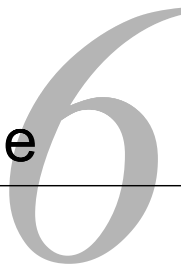
美国人口普查局提供各州之间人口流量的总结（从www.census.gov/hhes/migration/data/acs/state-to-state.html下载最新的XLS文件，并将其转换为CSV格式——在Excel或OpenOffice Calc中打开，并以CSV方式导出）。编写一个程序，给出在人口迁移的状态上最为相似的十对州。如果超过P_X/N个人从X移动到Y，则可以认为X和Y相似。其中$P_{X/Y}$是从X流出的总流量，N是州和地区的总数，而不是原始区域本身。尽量使用`numpy`进行数据存储和处理。每个相似对中的两个州是否通常都位于相同的海岸线？

我在巴黎艺术画廊见到了全世界最美的画。

——英国化学家和发明家Humphry Davy

第 6 章

使用series和frame



数据科学家往往青睐于表格数据（数组、矢量、矩阵）。表格数据具有很好的形式，可以方便地访问某个元素和某些行或列。超级计算机和许多现代高端个人电脑支持的矢量化算术运算，可以一次性作用在多个甚至所有表格项目上（第24单元给出了这种运算的numpy实现）。但是，numpy无法将数值数据本身与数据属性（比如列名、行名和索引）相绑定。正因为缺乏这样的参照物，使得同时使用多个numpy数组非常困难。

下面让我们进入pandas的世界。

pandas模块的初衷是为了给Python添加series和frames两个抽象的数据结构，它们其实是Python的竞争对手、最早的数据科学语言——R语言的核心。pandas以numpy为基础，对其进行了极大的扩展，并重新实现了部分功能。

pandas的frame本质上是一个“智能”电子表格：具有标签、列（变量）、行（观测记录），以及大量内置操作的表。（series是一个只有一列的frame。）表的数据部分（单元格）以numpy数组的方式实现。许多操作（例如数据变形和聚合，以及通用函数）也与numpy是类似的。通过行和列的标签，可以实现对行和列方便的、直截了当的访问。此外，标记的行和列允许pandas程序员（也就是我们）通过在“垂直”（堆叠）和“水平”（并排）方向上以合并和级联的方式来组合frame。在这个意义上，frame的工作方式很像关系数据库表。（请参阅第4章，回顾关系数据库的内容。）

最后，pandas可以很好地与pyplot集成在一起。pyplot是一个基于Python的绘图和数据可视化系统，具体内容将在第41单元介绍。坦率地讲，pandas具备开展数据科学研究所需要的一切。当然，也需要一些其他工具的配合。

本章通过第31单元对pandas的两个数据容器（Series和DataFrame）的介绍开启pandas的学习之旅。

第 31 单元

pandas 数据结构

pandas模块为已经具有丰富数据结构的Python增加了两个新的数据容器：Series和DataFrame。series是具有标签的（也就是具有索引的）一维矢量。frame是一个行和列都具有标签的表格，它与Excel电子表格和MySQL表格并无不同。frame的每一列都是一个series。除了一些特殊情况外，pandas对frame和series的处理方式是相似的。

frame和series不是简单的存储容器。它们都内置了进行各种数据整理操作的工具，如：

- 单级和分级索引
- 处理缺失数据
- 对整个列和表进行算术和布尔运算
- 数据库类型的操作（比如合并和聚合）
- 绘制各个列和整个表格
- 从文件读取数据并将数据写入文件

frame和series非常方便，当你在处理一维或二维表格数据时，都应该使用它们。

series

series是一维的数据矢量。和numpy数组一样（想要了解更多关于数组的内容，请参考第21单元），series是同构的：series的所有数据项必须是相同的数据类型。

可以使用任意序列（比如列表、元组或数组）创建一个简单的series。下面我们通过一组美国近期通货膨胀的数据，来展示pandas的series。首先需要说明的是，为了强调数据的不变性，我使用了一个元组来表示之前计算好的通货膨胀数据。另外，下面的例子并不需要经济学或财政学方面的高级知识！

```
import pandas as pd
# 最后的值是错误的，我们稍后再修改它！
inflation = pd.Series((2.2, 3.4, 2.8, 1.6, 2.3, 2.7, 3.4, 3.2, 2.8, 3.8, \
                       -0.4, 1.6, 3.2, 2.1, 1.5, 1.5))
```

```
⇒ 0     2.2
⇒ 1     3.4
⇒ 2     2.8
⇒ 3     1.6
⇒ 4     2.3
⇒ 5     2.7
⇒ 6     3.4
```

```
⇛ 7     3.2
⇛ 8     2.8
⇛ 9     3.8
⇛ 10   -0.4
⇛ 11    1.6
⇛ 12    3.2
⇛ 13    2.1
⇛ 14    1.5
⇛ 15    1.5
⇛ dtype: float64
```

函数`len()`是Python内置的标准通用函数，它也适用于series：

```
len(inflation)
```

```
⇛ 16
```

一个series，两个series，三个series……



series这个单词的单数形式和复数形式一样。它的由来可以追溯到拉丁语serere，其含义是**加入**或**连接**。

一个简单的series，比如刚刚创建的`inflation`，具有默认的整数索引：第一项的标签为0，第二项为1，以此类推。一个series的`values`属性包含了所有series值的列表；`index`属性指的是series的索引（索引是另一种`pandas`数据类型）；`index.values`属性指的是所有索引值组成的数组。

```
inflation.values
```

```
⇛ array([ 2.2,  3.4, 2.8, 1.6, 2.3, 2.7, 3.4, 3.2, 2.8,
⇛         3.8, -0.4, 1.6, 3.2, 2.1, 1.5, 1.5])
```

```
inflation.index
```

```
⇛ Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
⇛   dtype='int64')
```

```
inflation.index.values
```

```
⇛ array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
```

注意`pandas`是如何避免重复造轮子的，它使用`numpy`数组作为其底层存储工具！

所有这些数组（以及它们所代表的series属性）都是可变的，这一点有些令人惊讶。更改`values`、`index`和`index.values`实际上会改变series的值和索引。我们用这个事实来修改通货膨胀数据中的最后一项，把它改成一个错误的值：

```
inflation.values[-1] = 1.6
```

关于series存在这样一个问题：它看起来像一个数组，并且它的行为也和数组一样，于是根

据“鸭子测试”[1]的原理，series就是数组。例如，对于上面的例子，我们很难说明series的第一项对应的是哪一年的通胀数据。当然，你可以创建另一个series来保存年份，并保证两个series始终在一起使用，但是我不得不提醒读者，当我还在读高中的时候，就有人告诫我，像这样并列地使用series终将导致灾难。所以，让我们用一个自定义的索引来创建series，索引通过字典的方式传递给Series的构造函数。字典键作为series的索引，索引是series中不可分割的一部分：

```
inflation = pd.Series({1999 : 2.2, «more items», 2014 : 1.6, 2015 : np.nan})
```

```
⇒ 1999 2.2
⇒ «more items»
⇒ 2014 1.6
⇒ 2015 NaN
```

另外，也可以使用任意一个序列创建新的索引，然后将其附加到现有的series中：

```
inflation.index = pd.Index(range(1999, 2015))
inflation[2015] = numpy.nan
```

```
⇒ 1999 2.2
⇒ «more items»
⇒ 2014 1.6
⇒ 2015 NaN
```

series的值和索引可以有各自的名称，通过相应的name属性进行访问和分配。这些名称实际上起到了文档的作用，它们提醒我们（以及未来的核心读者）这两个序列的本质：

```
inflation.index.name = "Year"
inflation.name = "%"
```

```
⇒ Year
⇒ 1999 2.2
⇒ «more items»
⇒ 2014 1.6
⇒ 2015 NaN
⇒ Name: %, dtype: float64
```

通过在交互式命令行中打印series或者输入其名称，可以查看整个series的信息，包括所有的索引名称。也可以调用head()和tail()函数，这两个函数分别返回一个series的前五行和最后五行：

```
inflation.head()
```

```
⇒ Year
⇒ 1999 2.2
⇒ 2000 3.4
⇒ 2001 2.8
⇒ 2002 1.6
```

---

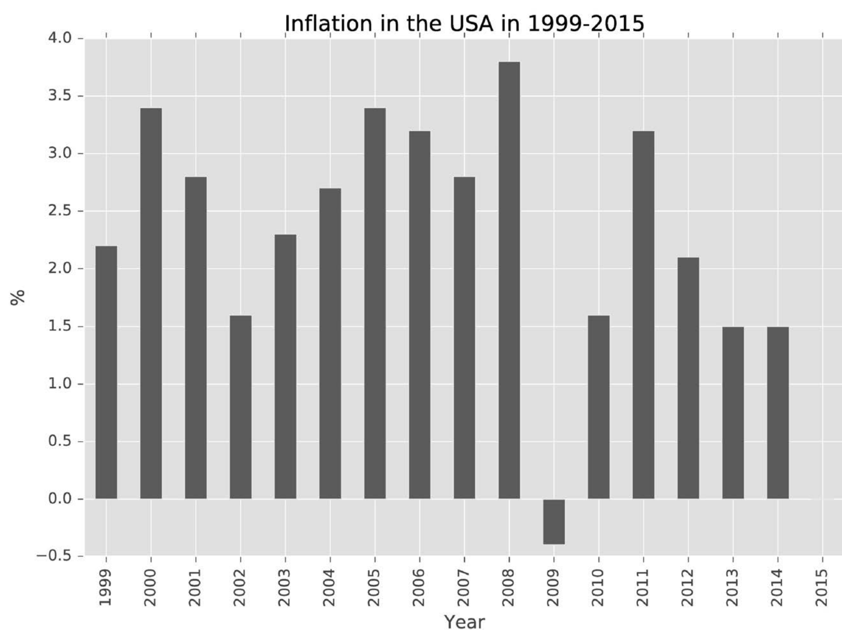
① 鸭子测试（Duck Test）是对一种归纳推理方式的幽默说法。它可以解释为：如果它看起来像鸭子，游泳像鸭子，叫声像鸭子，那么它可能就是只鸭子。——译者注

```
⇒ 2003 2.3
⇒ Name: %, dtype: float64

inflation.tail()

⇒ Year
⇒ 2011 3.2
⇒ 2012 2.1
⇒ 2013 1.5
⇒ 2014 1.6
⇒ 2015 NaN
⇒ Name: %, dtype: float64
```

与head()和tail()函数相比，也许你会更喜欢使用图片的方式（一图胜千言），可以参考下图（我们将在第8章进一步介绍绘图工具。）



series非常适合于观测一个变量的情况。但是，许多数据集都具有多个变量，这就轮到frame上场了。

frame

frame是一个行和列都具有标签的表。可以通过二维numpy数组、元组列表、Python字典或一个frame构造出一个新的frame。如果使用的是字典，则字典的键作为列的名称，字典的值（必须是序列）作为列值。如果使用的是frame，则pandas会将列名从源frame复制到新frame。如果使

用的是数组，可以通过可选参数`columns`（列名称组成的序列）提供列名称。沿用`numpy`的方式，frame索引为编号为0的轴（“垂直”维），frame列为编号为1的轴（“水平”维）。

我们使用美国酒精滥用和酒精中毒研究所[1]2011年公开的监测报告来研究frame。报告给出了1977—2009年美国每个州每年每种酒（啤酒、葡萄酒和烈酒）的人均消费量。

NIAAA报告



NIAAA报告是一个非常优秀的数据源，因此我将其经过预处理的一个副本作为了本书“代码”项，供读者浏览。不过要记住：不要喝酒，专心做数据科学！

可以将相同长度的行元组或其他序列的列表作为数据行传递给构造函数，来创建一个具有列名和索引的简单frame。（此处使用了第37单元的一个`pandas` CSV读取器，创建了完整的`alco` frame，然后选出了其中一年的数据）。

```
alco2009 = pd.DataFrame([(1.20, 0.22, 0.58),
                         (1.31, 0.54, 1.16),
                         (1.19, 0.38, 0.74),
                         «more rows»],
                        columns=("Beer", "Wine", "Spirits"),
                        index=("Alabama", "Alaska", «more states»))

⇒         Beer  Wine  Spirits
⇒ Alabama 1.20  0.22     0.58
⇒ Alaska  1.31  0.54     1.16
⇒ Arizona 1.19  0.38     0.74
⇒ «more rows»
```

你也可以使用列的字典，其结果与上面的代码相同：

```
alco2009 = pd.DataFrame({"Beer" :    (1.20, 1.31, 1.19, «more rows»),
                         "Wine" :    (0.22, 0.54, 0.38, «more rows»),
                         "Spirits" : (0.58, 1.16, 0.74, «more rows»)},
                        index=("Alabama", "Alaska", «more states»))
```

对于单个frame列的访问，可以使用字典或对象符号。但是，如果要添加新列，就必须使用字典符合。如果使用对象符号，则`pandas`会创建一个新的frame属性。和series一样，frame也有`head()`和`tail()`函数。（真的熊猫[2]也是有尾巴的！）

```
alco2009["Wine"].head()

⇒ State
⇒ Alabama       0.22
⇒ Alaska        0.54
```

① pubs.niaaa.nih.gov/publications/surveillance92/CONS09.pdf
② pandas也可译为熊猫。——译者注

```
⇒ Arizona       0.38
⇒ Arkansas      0.17
⇒ California    0.55
⇒ Name: Wine, dtype: float64
```

```
alco2009.Beer.tail()
```

```
⇒ State
⇒ Virginia         1.11
⇒ Washington       1.09
⇒ West Virginia    1.24
⇒ Wisconsin        1.49
⇒ Wyoming          1.45
⇒ Name: Beer, dtype: float64
```

和series一样，frame也支持广播：可以使用一条赋值语句给某列的所有行分配一个值。列甚至可以不存在；当列不存在时，pandas会自动创建它。

```
alco2009["Total"] = 0
alco2009.head()
```

```
⇒            Beer  Wine  Spirits  Total
⇒ State
⇒ Alabama    1.20  0.22     0.58      0
⇒ Alaska     1.31  0.54     1.16      0
⇒ Arizona    1.19  0.38     0.74      0
⇒ Arkansas   1.07  0.17     0.60      0
⇒ California 1.05  0.55     0.73      0
```

代码里对总量（Total）的设置显然是错误的，我们将在第36单元介绍如何用算术运算来修正这个值。

第 32 单元

数据重塑

数据标签是pandas对表格数据的主要贡献。所谓数据标签，是指与列和行相关联的数字或文本标签（例如与列对应的列名，以及与行对应的扁平化索引和分层索引）。这样的关联是非常灵活的：为了与其他frame相匹配，可以修改底层的numpy数组的形状（请参阅第22单元的reshape()函数），这种修改可能会使frame的某些行变为列，某些列变为行。例如，如果一个frame的分层索引具有两个级别（比如“Year”和“State”），而要匹配的另一个frame只具有“State”索引，则pandas会将“Year”标签自动转换为列名。在本单元中，你将了解扁平索引、分层索引和重建索引，以及其他重组数据标签的方式。

索引

frame的索引是一组分配给frame行的标签集合。（标签必须属于相同的数据类型,但标签的值不一定是唯一的。）通过可选参数`index`，可以给`DataFrame()`构造器提供索引。和series类似，可以通过属性`index.values`和`columns.values`访问和更改列名及索引。

```
alco2009.columns.values
```

```
⇒ array(['Beer', 'Wine', 'Spirits', 'Total'], dtype=object)
```

```
alco2009.index.values
```

```
⇒ array(['Alabama', 'Alaska', 'Arizona', «...»], dtype=object)
```

frame中的任意一列都可以作为一个索引——函数`reset_index()`和`set_index(column)`分别用于删除现有索引（如果存在的话）和建立一个新索引。这两个函数都返回一个新的frame，但是如果提供了可选参数`inplace=True`，则函数将直接修改frame对象本身：

```
alco2009.reset_index().set_index("Beer").head()
```

```
⇒            State  Wine  Spirits  Total
⇒ Beer
⇒ 1.20     Alabama  0.22     0.58      0
⇒ 1.31      Alaska  0.54     1.16      0
⇒ 1.19     Arizona  0.38     0.74      0
⇒ 1.07    Arkansas  0.17     0.60      0
⇒ 1.05  California  0.55     0.73      0
```

frame的索引是重要的行访问工具和相关的行标识符。无论使用哪个列作为索引，它必须是有意义的。在上面的例子中，使用的列就没有意义：啤酒消费量是州的属性，但不是标识符。

一旦有了索引，就可以通过行的索引属性`ix`来访问某一行，这就像是一个用索引标签作为键的行series的字典。frame的列作为每个series的索引：

```
alco2009.ix["Nebraska"]
```

```
⇒ Beer       1.46
⇒ Wine       0.20
⇒ Spirits    0.68
⇒ Total      0.00
⇒ Name: Nebraska, dtype: float64
```

Python运算符`in`可以检查具有某个标签的行是否存在于frame中：

```
"Samoa" in alco2009.index
```

```
⇒ False
```

函数`drop()`返回一个删除了一行或若干行（用列表表示）的frame的副本。如果要删除原始frame中的行，请传递可选参数`inplace=True`。

重建索引

重建索引从现有的frame或series中选择一定的行排列、列排列或者行和列排列，来创建新的frame或series。本质上，它等同于numpy的一个“智能”索引（参见第23单元），只不过如果pandas在原始frame中找不到请求的行或列标签，它将创建一个新的行或列，并用nan填充它（或它们）。

在下面的例子中，我们创建名称以“S”开头的州列表（该列表包含了不是州名的“Samoa”，当然它也不在alco2009 frame中）。然后，除了最后一列（即“Total”列，它没有被正确初始化）之外，再添加名为“Water”的列。最后，从原始的frame中提取所选的行和列。因为有一行和一列不在原始的frame alco 2009中，所以pandas会自动创建它们：

```
s_states = [state for state in alco2009.index if state[0] == 'S'] + ["Samoa"]
drinks = list(alco2009.columns) + ["Water"]
nan_alco = alco2009.reindex(s_states, columns=drinks)
```

```
⇒                 Beer  Wine  Spirits  Water
⇒ State
⇒ South Carolina 1.36  0.24     0.77    NaN
⇒ South Dakota   1.53  0.22     0.88    NaN
⇒ Samoa           NaN  NaN       NaN    NaN
```

可选参数method的可能值为"ffill"（正向填充）和"bfill"（后向填充），可以用于填补缺失的值。（这仅适用于单调减或单调增的索引。）关于数据插值的更多信息，请参阅第33单元。

分层索引

pandas支持分层（多级）索引和分层（多级）列名。多级索引也称为多重索引。

多级索引由三个列表组成：

- 级别名称
- 每个级别可能存在的所有标签
- frame或series中每一项的实际值的列表（列表的长度相同，并等于索引中的级别数）

下面的frame包含完整版本的NIAAA数据集，不只是2009年。它具有州名和年份的多级索引，并且按照两个索引进行排序：先按州名排序，后按年份排序。

```
alco
```

```
⇒               Beer Wine Spirits
⇒ State   Year
⇒ Alabama 1977 0.99 0.13    0.84
⇒         1978 0.98 0.12    0.88
⇒         1979 0.98 0.12    0.84
⇒         1980 0.96 0.16    0.74
⇒         1981 1.00 0.19    0.73
⇒ «...»
```

```
⇒ Wyoming 2005 1.21 0.23    0.97
⇒         2006 1.47 0.23    1.05
⇒         2007 1.49 0.23    1.10
⇒         2008 1.54 0.23    1.12
⇒         2009 1.45 0.22    1.10
```

数据转换操作通常会产生分层索引，但你也可以专门构建它们。函数`MultiIndex.from_tuples()`使用带有标签的元组和可选的级别名称列表生成多级索引。可以将多级索引附加到现有的frame或series中，或将其作为参数传递给构造函数`DataFrame()`：

```
multi = pd.MultiIndex.from_tuples((
        ("Alabama", 1977), ("Alabama", 1978), ("Alabama", 1979), ...,
        ("Wyoming", 2009)),
        names=["State", "Year"])
```

```
⇒ MultiIndex(levels=[['Alabama', 'Alaska', «...», 'Wyoming'],
⇒                    [1977, 1978, 1979, 1980, «...», 2009]],
⇒            labels=[[0, 0, 0, 0, 0, 0, 0, 0, «...», 50],
⇒                    [0, 1, 2, 3, 4, 5, 6, 7, «...», 32]],
⇒            names=['State', 'Year'])
```

```
alco.index = multi
```

可以使用与扁平化索引相同的多级索引。部分索引（使用几个标签中的某一个）产生一个frame；完整的索引产生一个series。

```
alco.ix['Wyoming'].head()
```

```
⇒      Beer Wine Spirits
⇒ Year
⇒ 1977 1.79 0.21    1.32
⇒ 1978 1.82 0.22    1.36
⇒ 1979 1.86 0.22    1.30
⇒ 1980 1.85 0.24    1.32
⇒ 1981 1.91 0.24    1.27
```

```
alco.ix['Wyoming', 1999]
```

```
⇒ Beer       1.41
⇒ Wine       0.18
⇒ Spirits    0.84
⇒ Name: (Wyoming, 1999), dtype: float64
```

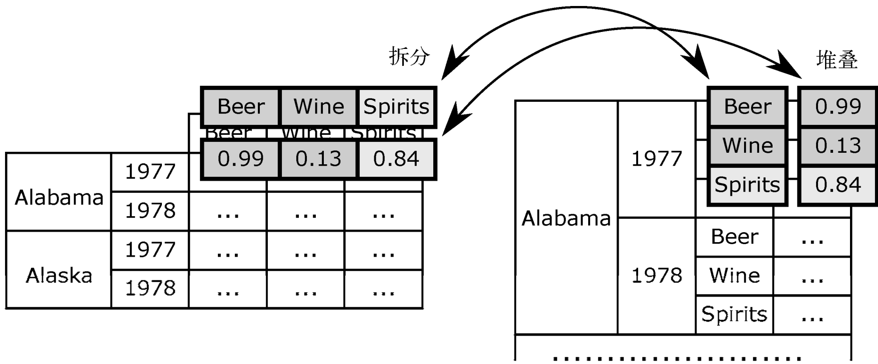
`pandas`使用相同的方式处理多级索引和列，这使得索引级别可能会变成列级别，反之亦然。

堆叠和旋转

使用多级的列名可以将多级索引完全或部分扁平化，同样，使用多级索引则可以将多级列名完全或部分扁平化。

`stack()`函数会增加索引的级别数，同时减少列名的级别数。它使得frame更高更窄，如下图

所示。如果列名已经扁平化了，则函数返回一个series。函数unstack()的作用相反：它减少索引的级别数，同时增加列名的级别数。它使得frame更短更宽。如果索引已经扁平化了，则函数返回一个series。




```
tall_alco = alco.stack()
tall_alco.index.names += ["Drink"]
tall_alco.head(10)
```

```
⇛ State    Year  Drink
⇛ Alabama  1977  Beer       0.99
⇛                Wine       0.13
⇛                Spirits    0.84
⇛          1978  Beer       0.98
⇛                Wine       0.12
⇛                Spirits    0.88
⇛          1979  Beer       0.98
⇛                Wine       0.12
⇛                Spirits    0.84
⇛          1980  Beer       0.96
⇛ dtype: float64
```

上述操作的结果是一个具有三级索引的series（在这里我不得不提供缺少的第三级的名称——“Drink”）。

```
wide_alco = alco.unstack()
wide_alco.head()
```

```
⇛            Beer                                                         ...
⇛ Year       1977  1978  1979  1980  1981  1982  1983  1984  1985  1986 ...
⇛ State                                                        ...
⇛ Alabama    0.99  0.98  0.98  0.96  1.00  1.00  1.01  1.02  1.06  1.09 ...
⇛ Alaska     1.19  1.39  1.50  1.55  1.71  1.75  1.76  1.73  1.68  1.68 ...
⇛ Arizona    1.70  1.77  1.86  1.69  1.78  1.74  1.62  1.57  1.67  1.77 ...
⇛ Arkansas   0.92  0.97  0.93  1.00  1.06  1.03  1.03  1.02  1.03  1.06 ...
⇛ California 1.31  1.36  1.42  1.42  1.43  1.37  1.37  1.38  1.32  1.36 ...

⇛ [5 rows x 99 columns]
```

上述操作的结果是具有扁平索引和两级分层列名的frame。你经常会在CSV和其他表格文件中看到这些类型的frame。为了使数据更加“方正”，更加易于管理，你可能需要将它们堆叠起来。

堆叠和拆分是更一般的**绕轴旋转**（pivoting）操作的特殊情况。函数`pivot(index, columns,values)`将frame转换为另一个frame，新的frame使用列索引作为新的索引，`columns`作为新的列名列表，`values`作为数据。

在下面的例子中，`alco`被重新组织成一个“方正”的frame，按年份（新的扁平索引）和州（列名）描述葡萄酒的消费情况：

```
alco.pivot("Year", "State", "Wine")

⇒ State Alabama Alaska Arizona Arkansas California Colorado Connecticut
⇒ Year
⇒ 1977     0.13   0.42    0.34     0.10       0.67     0.36        0.35
⇒ 1978     0.12   0.45    0.37     0.11       0.68     0.47        0.38
⇒ 1979     0.12   0.47    0.39     0.10       0.70     0.47        0.40
⇒ 1980     0.16   0.50    0.36     0.12       0.71     0.47        0.43
⇒ «...»

⇒ [33 rows x 51 columns]
```

如果`index`为`None`，则`pandas`会重新使用原始frame的索引。

第 33 单元

处理缺失数据

数据几乎从来都不是完美的。有些值是固定的（不需要担心这样的值），有些值是有问题的（你必须用加盐[①]的方式对待它们），而有些值是缺失的。

`pandas`沿用了传统的`numpy.nan`（第25单元）来表示缺失数据，这样做的好处是不会将缺失数据与其他任何数字混淆，因为它的名称类似于R语言中的`NA`（“Not Available”，不可用）符号。`pandas`还提供识别和插补缺失值的函数。

造成series和frame中出现缺失值的原因有以下几个：数据可能从未被收集；数据被收集了，但由于是不恰当的值而被丢弃了；你将几个本身是完整的数据集组合起来了，但是它们的组合却不再是完整的数据集。可惜的是，在处理缺失值之前，不能进行任何严谨的数据分析。缺失值必须被删除或插补。所谓插补就是将其替换为有意义的值。下面进行详细介绍。

① “加盐”是数据加密中常用的术语，具体来说就是在原有材料（用户自定义密码）中加入其他成分（一般是用户自有且不变的因素），以此来增加系统复杂度。——译者注

删除缺失数据

处理缺失数据最简单的方式是假装你从来没有得到它（“视而不见，充耳不闻”的方法）。dropna()函数可以删除部分（how="any"，默认值）或者全部（how="all"）无效的列（axis=0，默认值）或行（axis=1），并返回“干净的”frame对象。可以使用可选参数inplace=True直接修改原始frame，而不是创建副本。

```
nan_alco.dropna(how="all")
```

```
⇒                 Beer Wine Spirits Water
⇒ State
⇒ South Carolina 1.36 0.24    0.77   NaN
⇒ South Dakota   1.53 0.22    0.88   NaN
```

```
nan_alco.dropna(how="all", axis=1)
```

```
⇒                 Beer  Wine  Spirits
⇒ State
⇒ South Carolina 1.36  0.24     0.77
⇒ South Dakota   1.53  0.22     0.88
⇒ Samoa           NaN   NaN      NaN
```

只删除缺失值而不破坏frame的结构是不可能的。只能对包含“脏的”单元格的整行或整列进行删除，并且完成删除操作后“干净的”frame可能是空的，这样一来什么数据都看不到了。

```
nan_alco.dropna()
```

```
⇒ Empty DataFrame
⇒ Columns: [Beer, Wine, Spirits, Water]
⇒ Index: []
```

6

插补缺失数据

处理缺失值的另一种方法是插补缺失数据。插补缺失值意味着用一些有意义的、“干净的”值代替它们。当然，什么是有意义的取决于数据本身。只有数据科学家才能判断替代值是否合适。

两种最常见的插补技术是用常数（0、1等）和“干净”值的平均值替换缺失值。但首先，需要明确哪些是缺失值。

函数isnull()和notnull()是互补的。当某个值是nan时，函数isnull()返回True。当某个值不是nan时，函数notnull()返回True。请注意，根据IEEE 754的浮点数标准，表达式np.nan==np.nan为False，这使得直接比较是不可能的！

```
nan_alco.isnull()
```

```
⇒                 Beer  Wine  Spirits  Water
⇒ State
⇒ South Carolina False False    False   True
```

```
⇒ South Dakota   False False   False   True
⇒ Samoa           True  True    True   True
```

```
nan_alco.notnull()
```

```
⇒                Beer   Wine   Spirits   Water
⇒ State
⇒ South Carolina True   True      True   False
⇒ South Dakota   True   True      True   False
⇒ Samoa         False  False     False   False
```

下面我们通过估算平均值来修正“Spirits”列（请注意，在numpy中，连字符“-”是否定运算符）：

```
sp = nan_alco['Spirits'] # 选中具有“脏”值的一列数据
clean = sp.notnull() # “干净”值的行编号
sp[-clean] = sp[clean].mean() # 使用平均值来修正“脏”值
nan_alco
```

```
⇒                Beer  Wine  Spirits  Water
⇒ State
⇒ South Carolina 1.36  0.24    0.770    NaN
⇒ South Dakota   1.53  0.22    0.880    NaN
⇒ Samoa           NaN   NaN    0.825    NaN
```

平均值插补的方法必须逐列（或逐行）地进行，但如果是常数插补，就可以直接应用到整个frame。函数fillna(val)是将常数val插到空缺处的最简单的方式。另外，该函数可以沿着列（axis=0，默认值）或行（axis=1），将最后一次有效观测值向前（method="ffill"）或向后（method="bfill"）复制。除非指定了参数inplace=True，否则该函数将返回一个新的frame或series。

```
nan_alco.fillna(0)
```

```
⇒                Beer  Wine  Spirits  Water
⇒ State
⇒ South Carolina 1.36  0.24     0.77      0
⇒ South Dakota   1.53  0.22     0.88      0
⇒ Samoa          0.00  0.00     0.00      0
```

```
nan_alco.fillna(method="ffill")
```

```
⇒                Beer  Wine  Spirits  Water
⇒ State
⇒ South Carolina 1.36  0.24     0.77    NaN
⇒ South Dakota   1.53  0.22     0.88    NaN
⇒ Samoa          1.53  0.22     0.88    NaN
```

替换数据

另一种处理特定的“脏”数据的方法，是根据具体情况使用“干净”值有选择地替换它们。

replace(val_or_list,new_val)函数将一个值或值列表替换成一个新的值或值列表，列表的长度必须相同。除非传递inplace=True参数，否则该函数会返回一个新的frame或series。

函数combine_first(pegs)将两个frame或两个series组合在一起。它用frame/series参数中的相应值替换frame/series对象中的缺失值。从某种意义上来说，函数的参数提供了默认值。

第 34 单元 组合数据

使用series或frame时，有用的数据可能存在于多个frame中，因此可能需要组合数据以备进一步处理。pandas提供了合并和连接frame的函数——你只需要判断是否需要合并或连接。

合并

frame的合并类似于数据库表的合并：pandas将左侧和右侧frame中具有相同索引（或其他指定列中具有相同值）的行组合在一起。如果左侧frame与右侧frame中相匹配的行是唯一的，则称该类型的合并为**一对一合并**。当有多个匹配时，该类型的合并称为**一对多合并**。对于一对多合并而言，pandas会根据需要复制左侧frame的行，而复制可能会导致行的重复（我们将在下一小节介绍如何处理重复的行）。当两个frame中都有多行相匹配时，这种合并类型称为**多对多合并**，而pandas还是会根据需要进行行复制，并将numpy.nans插入缺失数据的单元。

6

如果两个frame具有一个名称相同的列（即键列），就可以将该列上的frame合并。即使没有这样的列，也可以指定其他列作为键，如下所示：

```
df = pd.merge(df1, df2, on="key")
df = pd.merge(df1, df2, left_on="key1", right_on="key2")
```

使用美国人口普查局截至2009年7月1日的数据[①]，建立一个关于美国人口的frame。除了包含每个州的人口数据，该frame还包含美国东部、东北部、西北部、中西部、西部和南部，以及整个美国的人口数据。

```
population.head()
```

```
⇒                           Population
⇒ State
⇒ Wyoming                       544270
⇒ District of Columbia          599657
⇒ Vermont                       621760
⇒ North Dakota                  646844
⇒ Alaska                        698473
```

① www.census.gov/popest/data/historical/2000s/vintage_2009/state.html

population和alco2009这两个frame都具有“State”索引，因此可以将索引删除，把两个frame在公共列上的所有内容合并，同时观察酒的消费量和人口数量：

```
df = pd.merge(alco2009.reset_index(),
              population.reset_index()).set_index("State")
df.head()
```

```
⇒             Beer  Wine  Spirits  Population
⇒ State
⇒ Alabama     1.20  0.22     0.58     4708708
⇒ Alaska      1.31  0.54     1.16      698473
⇒ Arizona     1.19  0.38     0.74     6595778
⇒ Arkansas    1.07  0.17     0.60     2889450
⇒ California  1.05  0.55     0.73    36961664
```

如果两个frame中有相同名称的列，则pandas会将后缀"_l"和"_r"添加到列名称中。使用可选参数suffixes（两个字符串组成的元组）控制后缀。如果要在索引列而不是一般的列上合并，请使用可选参数left_index=True和right_index=True（可以同时使用或单独使用）。以下语句的结果与之前的相同，但默认的排序顺序可能存在差异：

```
df = pd.merge(alco2009, population, left_index=True, right_index=True)
df.head()
```

```
⇒                       Beer  Wine  Spirits  Population
⇒ State
⇒ Wyoming               1.45  0.22     1.10      544270
⇒ District of Columbia  1.26  1.00     1.64      599657
⇒ Vermont               1.36  0.63     0.70      621760
⇒ North Dakota          1.63  0.25     1.16      646844
⇒ Alaska                1.31  0.54     1.16      698473
```

如果两个索引都是键，就要使用join()函数替换上述的merge()函数：

```
population.join(alco2009).tail(10)
```

```
⇒               Population  Beer  Wine  Spirits
⇒ State
⇒ Illinois        12910409  1.22  0.39     0.73
⇒ Florida         18537969  1.21  0.48     0.92
⇒ New York        19541453  0.91  0.46     0.69
⇒ Texas           24782302  1.42  0.28     0.58
⇒ California      36961664  1.05  0.55     0.73
⇒ Northeast       55283679   NaN   NaN      NaN
⇒ Midwest         66836911   NaN   NaN      NaN
⇒ West            71568081   NaN   NaN      NaN
⇒ South          113317879   NaN   NaN      NaN
⇒ United States 307006550   NaN   NaN      NaN
```

join()和merge()这两个函数使用同一个可选参数how，它的可接受值为"left"（join()函数的默认值）、"right"、"inner"（merge()函数的默认值）或"outer"。左连接使用调用函数的frame（即左侧frame）的索引。右连接使用作为参数的frame（即右侧frame）的索引。外连

接使用索引的并集。内连接使用索引的交集。（pandas的连接类型与先前在第18单元中介绍的MySQL连接类型一致。）

如果frame的索引不相同，则左连接（或合并）、右连接（或合并）和外连接（或合并）操作将引入具有缺失值的行。在上述例子中，这种情况发生在啤酒、葡萄酒和烈酒的总量未知的地方。内连接（或合并）不会引入新的缺失值。

如果两个frame具有名称相同的列，则必须提供可选参数lsuffix和rsuffix（这两个参数都是字符串）。pandas会将后缀附加到公共的列名称之后。

join()函数还可以使用可选参数on，实现在共享的列名称上连接两个frame（但不能用名称不同的列，或者一个是列而另一个是索引）。

连接

concat()函数通过将一组frame彼此“竖直”（axis=0，默认值）或“水平”（axis=1）地放置在一起来连接它们，并返回一个新的frame：

```
pd.concat([alco2009, population], axis=1).tail()
```

```
⇛                Beer  Wine  Spirits  Population
⇛ Washington     1.09  0.51     0.74     6664195
⇛ West            NaN   NaN      NaN    71568081
⇛ West Virginia  1.24  0.10     0.45     1819777
⇛ Wisconsin      1.49  0.31     1.16     5654774
⇛ Wyoming        1.45  0.22     1.10      544270
```


如果frame的维度不匹配，pandas就会在所谓的“空位”处加入具有缺失值的行或列。

pandas忠实地保留所有竖直堆叠的frame的索引，这可能导致索引出现重复的键。这并不妨碍你的使用，当然你也可以选择删除重复的项（具体方法将在下一小节介绍），或者通过可选参数keys（字符串列表），给新的frame添加二级索引，从而形成分层索引。对于“水平”方向的（逐列的）连接，可以使用相同的参数创建分层列名。

我们使用加拿大统计局网站[①]的数据，创建一个关于2011年加拿大省份人口的frame。接下来我们就可以创建一个新的frame来描述北美国家人口，结合美国和加拿大的两个人口的frame，可以给出一个合适的新索引。（请注意，美国的frame比加拿大的早两年。）

```
pop_na = pd.concat([population, pop_ca], keys=["US", "CA"])
pop_na.index.names = ("Country", "State")
```

```
⇛                        Population
⇛ Country State
⇛ US      Wyoming            544270
```

① www.statcan.gc.ca/tables-tableaux/sum-som/l01/cst01/demo02a-eng.htm

```
⇒         District of Columbia  599657
⇒         Vermont               621760
⇒         North Dakota          646844
⇒         Alaska                698473
⇒ «...»
⇒ CA      Alberta               3790200
⇒         British Columbia      4499100
⇒         Yukon                   35400
⇒         Northwest Territories   43500
⇒         Nunavut                 34200
```

请记住，必要时，分层索引可以被扁平化。

两个frame应该进行合并还是连接呢？



merge()和concat()都可以将两个或多个frame组合在一起。concat()用于组合具有相似内容的frame（比如美国各州和加拿大各省的人口："苹果到苹果"），而merge()则用于组合具有互补内容的frame（比如人口和酒精消费率："苹果到橘子"）。

删除重复行

函数duplicated([subset])返回一个布尔型的series，表示在所有列或subset表示的列中是否有重复的行（subset是一个列名称组成的序列）。可选参数keep用于控制标记的重复行是第一个重复行（"first"）、最后一个重复行（"last"），还是每个（True）重复行都标记。

函数drop_duplicates()返回一个删除了所有列或subset表示的列中重复行的frame或series（subset是一个列名称组成的序列）。可选参数keep用于控制删除的行是第一个重复行（"frist"）、最后一个重复行（"last"），还是每个（True）重复行都删除。使用可选参数inplace=True可以从原始对象中删除重复项。

第35单元

数据的排序和描述

使用frame来表示数据是远远不够的。接下来我们需要一个对数据进行排序和描述的标尺。Python有通用的标尺函数len()以及该函数的"兄弟函数"min()和max()，这几个函数是不错的出发点。但是除了**多少个**、**多少钱**这样的问题外，我们经常需要知道更多问题的答案。pandas提供了许多函数，用于排序、分级、计数、会员测试和获取描述性的统计信息。

排序和分级

series和frame可以按索引或值进行排序。函数sort_index()返回按索引排序的frame（该函数不适用于series）。排序顺序总是字典序（数字按数值大小排序，字符串按字母表的顺序排序），可以使用参数ascending（默认值为True）来控制升序或降序。和之前一样，选项inplace=True可以使pandas对原始的frame进行排序。

```
population.sort_index().head()
```

```
⇒             Population
⇒ State
⇒ Alabama        4708708
⇒ Alaska          698473
⇒ Arizona        6595778
⇒ Arkansas       2889450
⇒ California    36961664
```

函数sort_values()返回一个按值排序的frame或series。对于frame来说，第一个参数是某一列的名称或者一组列名称构成的列表，对应的可选参数ascending可以是布尔值或布尔值构成的列表（与要排序的数据列一一对应）。参数na_position（取值为"first"或"last"）指定值为nan的单元在排序后的位置（在开头或结尾）。

```
population.sort_values("Population").head()
```

```
⇒                       Population
⇒ State
⇒ Wyoming                   544270
⇒ District of Columbia      599657
⇒ Vermont                   621760
⇒ North Dakota              646844
⇒ Alaska                    698473
```

你知道怀俄明州（Wyoming）是美国人口最少的州吗？现在经过这一番数据分析后，这个事实就很清楚了。

函数rank()为frame或series的每个值计算一个数值等级。如果有相等的值，则该函数会给它们分配一个平均的等级。布尔型参数numeric_only仅将等级限制为数值。参数na_option（取值为"top"、"bottom"或"keep"）设定nan的处理方式：将它们移动到新frame的顶部或底部，又或者将它们保留在原始的frame中。

```
pop_by_state = population.sort_index()
pop_by_state.rank().head()
```

```
⇒             Population
⇒ State
⇒ Alabama             29
⇒ Alaska               5
⇒ Arizona             38
⇒ Arkansas            20
⇒ California          51
```

6

现在你只需要敲十几个按键来将上面的输出与原始的人口frame相合并或连接，就可以得到一个包含实际人口以及州的排名的frame。

描述性统计量

描述性统计函数计算一个series或frame的每一列的和（sum()）、均值（mean()）、中值（median()）、标准差（std()）、数量（count()）、最小值（min()）和最大值（max()）。这些函数都可以使用布尔型参数skipna，它控制是否从分析中排除nan值，而参数axis告诉函数以哪种方向获取数据（“垂直”或“水平”）。

```
alco2009.max()

⇒ Beer       1.72
⇒ Wine       1.00
⇒ Spirits    1.82
⇒ dtype: float64
```

```
alco2009.min(axis=1)

⇒ State
⇒ Alabama    0.22
⇒ Alaska     0.54
⇒ Arizona    0.38
⇒ Arkansas   0.17
⇒ California 0.55
⇒ dtype: float64
```

```
alco2009.sum()

⇒ Beer       63.22
⇒ Wine       19.59
⇒ Spirits    41.81
⇒ dtype: float64
```

函数argmax()（针对series）和idxmax()（针对frame）找出最大值首次出现的索引位置。这两个函数很容易记住：它们是pandas在处理series和frame时仅有的没有统一的两个函数。

pandas对伪积分、伪微分和其他累积方法的支持比较有限。函数cumsum()、cumprod()、cummin()和cummax()分别计算从series或frame的每列中的第一项开始的累积和、累积乘积、累积最小值和累积最大值。可以使用cumsum()函数获得夏威夷（或其他任意一个州）的累积酒精消费量：

```
alco.ix['Hawaii'].cumsum().head()

⇒      Beer  Wine  Spirits  Total
⇒ Year
⇒ 1977 1.61  0.36     1.26   3.23
⇒ 1978 2.99  0.82     2.56   6.37
```

```
⇒ 1979 4.59  1.26     3.84   9.69
⇒ 1980 6.24  1.72     5.05  13.01
⇒ 1981 7.98  2.16     6.21  16.35
```

函数`diff()`计算连续的列或series项之间的滑动差。计算结果的第一行是没有定义的。使用`diff()`函数也可以找出夏威夷每年酒消费量的变化情况：

```
alco.ix['Hawaii'].diff().head()
```

```
⇒        Beer   Wine  Spirits         Total
⇒ Year
⇒ 1977    NaN    NaN      NaN           NaN
⇒ 1978 -0.23   0.10     0.04  -9.000000e-02
⇒ 1979  0.22  -0.02    -0.02   1.800000e-01
⇒ 1980  0.05   0.02    -0.07  -4.440892e-16
⇒ 1981  0.09  -0.02    -0.05   2.000000e-02
```

执行伪微分操作之后，计算结果中最后一列的名称具有误导性。可以将其改为“Δ(Total)”“Change of Total”等，以免混淆。

唯一性、计数、会员资格

`numpy`可以将数组视为集合（如第28单元所述）。`pandas`也可以将series作为集合来处理（但frame是不能这样处理的）。下面我们再次使用第28单元中伪生物信息学的例子，来认真地练习我们的series集合技能：

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna_as_series = pd.Series(list(dna), name="genes")
dna_as_series.head()
```

```
⇒ 0    A
⇒ 1    G
⇒ 2    T
⇒ 3    C
⇒ 4    C
⇒ Name: genes, dtype: object
```

函数`unique()`和`value_counts()`分别算出series和frame中不同值组成的数组，并能给出每个不同值出现的次数（可以将其与第7单元的计数器`Counter`相比较）。如果series中包含`nan`，则它们同样会被计数。

```
dna_as_series.unique()
```

```
⇒ array(['A', 'G', 'T', 'C'], dtype=object)
```

```
dna_as_series.value_counts().sort_index()
```

```
⇒ A    5
⇒ C    6
⇒ G    7
```

```
⇒ T    4
⇒ Name: genes, dtype: int64
```

isin()函数是专门为pandas的两个主要数据类型（即series和frame）定义的。它返回一个与数据大小相同的布尔型的series或frame，用于确定series或frame的每一项是否存在于某个集合中。我们知道，在DNA序列中只有核苷酸A、C、G和T。下面的代码可以告诉我们，之前构造出的DNA序列中**所有的**核苷酸是否都有效。

```
valid_nucs = list("ACGT")
dna_as_series.isin(valid_nucs).all()
```

```
⇒ True
```

至此，你应该对数据感到非常满意了，或许你已经等不及要进行一些数值处理，以期获得很棒的结果。接下来我们就来介绍数据转换的工具。

第 36 单元

数据转换

你现在已经准备好一睹pandas的“强劲套装”了，它们是矢量化的算术、逻辑运算以及其他数据转换机制。

算术运算

pandas支持四种算术运算（加法、减法、乘法和除法）和numpy的通用函数（ufunc，参考第25单元）。可以使用相应的运算符和函数来组合具有相同大小和结构的frame、frame列和series，以及相同大小的series。

有了算术运算，我们终于可以来更正alco frame的“Total”列了：

```
alco["Total"] = alco.Wine + alco.Spirits + alco.Beer
alco.head()
```

```
⇒                Beer  Wine  Spirits  Total
⇒ State   Year
⇒ Alabama 1977 0.99  0.13     0.84   1.96
⇒         1978 0.98  0.12     0.88   1.98
⇒         1979 0.98  0.12     0.84   1.94
⇒         1980 0.96  0.16     0.74   1.86
⇒         1981 1.00  0.19     0.73   1.92
```

如果要用对数尺度来表示总消耗量，可以使用numpy提供的log10()、log()和许多其他通用函数：

```
np.log10(alco.Total).head()
```

```
⇒ State     Year
⇒ Alabama  1977  0.292256
⇒          1978  0.296665
⇒          1979  0.287802
⇒          1980  0.269513
⇒          1981  0.283301
⇒ Name: Total, dtype: float64
```

所有算术运算都保留索引。该功能称为**数据对齐**：两个series相加时，pandas会将一个series中带索引“C”的项添加到另一个series中的同名项目中。如果不存在同名项目，则相加的结果为nan。下面我们来模拟基因工程的做法，从第35单元第3小节的原始DNA片段中去除核苷酸C和T，得出两个DNA片段，然后对两个片段的不同核苷酸计数：

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna1 = dna.replace("C", "")
dna2 = dna.replace("T", "")
dna_as_series1 = pd.Series(list(dna1), name="genes") # 移除所有C
dna_as_series2 = pd.Series(list(dna2), name="genes") # 移除所有T
dna_as_series1.value_counts() + dna_as_series2.value_counts()
```

```
⇒ A    10
⇒ C   NaN
⇒ G    14
⇒ T   NaN
⇒ Name: genes, dtype: float64
```

在进行数据聚合之前，检查一下数据，看看是否需要进行缺失数据的清理（参考第33单元）。

6

数据聚合

数据聚合由数据分割、聚合和组合这三个步骤组成。

(1) 在分割步骤中，数据按单个键或多个键分割成块 。

(2) 在聚合函数的应用步骤中，将一个聚合函数（比如sum()或count()）应用于每个数据块。

(3) 在组合步骤中，计算结果被合并为一个新的series或frame。

pandas的强大就在于它拥有groupby()函数和大批聚合函数，可以自动执行这里列出的三个步骤，所以我们要做的就是喝杯咖啡、坐享其成。

函数groupby()通过基于一个或多个分类的键值将行分到不同的组中，从而实现对frame的分割。该函数返回一个组生成器，可以在循环中使用（来访问组的内容）或者与聚合函数一起使用。

聚合函数包括count()（返回组中的行数），sum()（返回组中数字行的总和），mean()、median()、std()和var()（每个函数都返回组中所有行相应的统计量度量），min()和max()（返回组中最小和最大的行），prod()（返回组中数字行的乘积），first()和last()（返回组中的

第一行和最后一行，该函数仅对有序的frame有意义）。

下面来考察一下我们一直在使用的alco frame，并得出一年中所有州总的酒消费量：

```
# 下面按年份（列“Year”）分组
alco_noidx = alco.reset_index()
sum_alco = alco_noidx.groupby("Year").sum()
sum_alco.tail()
```

```
⇒        Beer   Wine   Spirits   Total
⇒ Year
⇒ 2005 63.49  18.06    38.89   120.44
⇒ 2006 64.37  18.66    40.15   123.18
⇒ 2007 64.67  19.08    40.97   124.72
⇒ 2008 64.67  19.41    41.59   125.67
⇒ 2009 63.22  19.59    41.81   124.62
```

如果使用多个列进行分割，则结果具有多个索引，每个列对应一个级别的索引。

还可以在一个for循环中，通过组的迭代访问每个组的内容。在每次迭代中，groupby()函数返回的生成器提供索引条目以及与条目对应的行组（以frame的形式）：

```
for year, year_frame in alco_noidx.groupby("Year"):
   «do_something(year, year_frame)»
```

有时你可能希望使用一个可计算的属性来实现行的分组，而不是使用现有的某个列或多个列。pandas允许使用字典或series的映射来实现数据聚合。让我们考虑一个将州映射到美国人口普查局定义的地区[①]的字典：

```
state2reg
```

```
⇒ {'Idaho': 'West', 'West Virginia': 'South', 'Vermont': 'Northeast', «...»}
```

现在可以按地区计算酒精的平均消费量了！请记住，字典的操作对象是行索引标签，而不是行的值，因此需要给某些列分配frame索引（至少在分组操作时需要这样处理）。字典的值（同时也是组的名称）成为返回的frame的索引：

```
alco2009.groupby(state2reg).mean()
```

```
⇒                Beer       Wine    Spirits
⇒ Midwest   1.324167   0.265000   0.822500
⇒ Northeast 1.167778   0.542222   0.904444
⇒ South     1.207500   0.275625   0.699375
⇒ West      1.249231   0.470769   0.843846
```

奥卡姆[②]是英国方济会的一位修道士、经院哲学家和神学家。用他的话说（其实并不是他说的），**数据聚合精简了实体**，因此对我们是有好处的。相反，离散化将值转换为类别，**增加了实**

① www2.census.gov/geo/docs/maps-data/maps/reg_div.txt
② en.wikipedia.org/wiki/William_of_Ockham

体，这对我们是不利的，除非它确实非常有用。

离散化

离散化是指将连续变量转换为离散（**分类**）变量，通常用于直方图和机器学习（参见第10章）。

函数`cut()`将作为第一个参数的数组或series分割成半开半闭的区间（即类别）。第二个参数是一个数（代表相同大小的区间的个数），或者是一个区间的边界列表。如果要将序列分割成N个区间，则可以传递N+1个区间边界组成的列表。由`cut()`函数生成的类别属于序数型数据：可以对它们进行排序和相互比较。

```
cats = pd.cut(alco2009['Wine'], 3).head()
```

```
⇒ State
⇒ Alabama       (0.0991, 0.4]
⇒ Alaska           (0.4, 0.7]
⇒ Arizona       (0.0991, 0.4]
⇒ Arkansas      (0.0991, 0.4]
⇒ California       (0.4, 0.7]
⇒ Name: Wine, dtype: category
⇒ Categories (3, object): [(0.0991, 0.4] < (0.4, 0.7] < (0.7, 1]]
```

如果你想制作自己的类别标签，只需要传递另一个可选参数`labels`（N个标签组成的列表，每个区间对应一个标签）。

```
cats = pd.cut(alco2009['Wine'], 3, labels=("Low", "Moderate", "Heavy"'))
cats.head()
```

```
⇒ State
⇒ Alabama            Low
⇒ Alaska        Moderate
⇒ Arizona            Low
⇒ Arkansas           Low
⇒ California    Moderate
⇒ Name: Wine, dtype: category
⇒ Categories (3, object): [Low < Moderate < Heavy]
```

如果设置`labels=False`，则`cut()`函数只对数据区间进行编号而不做标记，并返回区间的成员信息：

```
cats = pd.cut(alco2009['Wine'], 3, labels=False).head()
```

```
⇒ State
⇒ Alabama       0
⇒ Alaska        1
⇒ Arizona       0
⇒ Arkansas      0
⇒ California    1
⇒ Name: Wine, dtype: int64
```

6

函数qcuts()与函数cuts()类似，不同的是qcuts()使用分位数而不是区间宽度进行分割。可以用它来计算分位数（比如中位数和四分位数）。

```
quants = pd.qcut(alco2009['Wine'], 3, labels=("Low", "Moderate", "Heavy"))
quants.head()
```

```
⇒ State
⇒ Alabama          Low
⇒ Alaska         Heavy
⇒ Arizona     Moderate
⇒ Arkansas         Low
⇒ California     Heavy
⇒ Name: Wine, dtype: category
⇒ Categories (3, object): [Low < Moderate < Heavy]
```

另一种使用少量可能值（这些值已分过类！）离散化变量的方法是将其分解为一组虚拟**指标**变量，每个变量对应一个可能的值。

虚拟变量是一个布尔型变量，与其对应的类别变量的值为true，其他所有值为false。在逻辑回归以及其他形式的机器学习中用到了虚拟变量，第10章将讨论相关内容。

函数get_dummies()将数组、series或frame转换为与原始对象拥有相同索引的另一个frame，每个列对应一个虚拟变量。如果对象是一个frame，请使用可选参数columns（需要离散化的列组成的列表）。

回顾在第36单元第2小节给出的州的地区划分，此处给出这种划分的使用指示器的表示法：

```
pd.get_dummies(state2reg).sort_index().head()
```

```
              Midwest  Northeast  South  West
⇒ state
⇒ Alabama           0          0      1     0
⇒ Alaska            0          0      0     1
⇒ Arizona           0          0      0     1
⇒ Arkansas          0          0      1     0
⇒ California        0          0      0     1
```

因为每个州都只属于一个区域，所以每行中所有值的总和总是等于1，对于任何虚拟变量也是如此。

映射

映射是最一般的数据转换方式。它使用map()函数将任意的单参数函数应用于选中列中的每个元素。单参数函数可以是内置的Python函数、任意导入模块的函数、用户定义的函数或匿名的lambda函数。

举个例子，我们来创建三个字母的州名缩写。

```
with_state = alco2009.reset_index()
abbrevs = with_state["State"].map(lambda x: x[:3].upper())
abbrevs.head()
```

```
⇒ 0    ALA
⇒ 1    ALA
⇒ 2    ARI
⇒ 3    ARK
⇒ 4    CAL
⇒ Name: State, dtype: object
```

显然，我们没能创建出唯一的三个字母的缩写，但为了理解`map()`函数，这个例子还是值得一试的！

与高度优化和并行化的通用函数不同，作为参数传递给`map()`的函数是由Python解释器执行的，无法对其进行优化。这使得`map()`函数非常低效，所以应该在没有其他选择时才使用它。

交叉表

交叉表计算组频率，并返回一个frame，其行和列分别对应两个分类变量（因子）的不同值。如果提供可选参数`marginins=True`，该函数还会计算行和列小计值。

以下代码计算一个州是“葡萄酒州”（葡萄酒消费量高于平均水平）还是“啤酒州”（啤酒消费量高于平均水平）的联合频率：

```
wine_state = alco2009["Wine"] > alco2009["Wine"].mean()
beer_state = alco2009["Beer"] > alco2009["Beer"].mean()
pd.crosstab(wine_state, beer_state)
```

```
⇒ Beer   False True
⇒ Wine
⇒ False     14   15
⇒ True      12   10
```

如果表中数字的差异不是很大（本例中的数字差异就不大！），那么这两个因子可能就是独立的。我们将在第47单元第2小节中回顾这个例子。

6

第 37 单元　掌握 pandas 的文件读写功能

即使你想不出使用pandas的理由，你仍然会被pandas的文件读写（即输入和输出）功能所折服。pandas的输入和输出功能一方面可以实现frame和series之间的数据交换，另一方面能实现CSV文件、表格文件、固定宽度文件、JSON文件（在第15单元已经讨论过）、操作系统剪贴板等

之间的数据交换。pandas支持：

- 自动索引和列名提取
- 数据类型推断、数据转换和缺失数据的检测
- 日期时间解析
- 消除“不干净的”数据（跳过行、页脚和评论；处理数千个分隔符）
- 数据分块

读取 CSV 和表格文件

函数read_csv()根据文件名或打开的文件句柄，从指定的CSV文件中读取一个frame。该函数具有近五十个可选参数，它是处理CSV文件的瑞士军刀。我们自然不会再使用CSV读取器reader()了（参阅第14单元，也许现在可以抛弃它们了）。read_csv()函数的部分重要参数如下。

- sep或delimiter：列分隔符。read_csv()的这个参数可以接受正则表达式（例如r"\s+"表示的“任意多个空白”）。
- header：作为列名的行号。如果你有自己的列名列表，则传递None。
- index_col：作为索引的列名。如果传递False，则pandas将生成一个默认数字索引。
- skiprows：要跳过的文件头行数或行号列表。
- thousands：表示大数时使用的千位分隔符。
- names：列名列表。
- na_values：用于处理缺失数据的字符串或字符串列表。如果要为不同的列使用不同的字符串（例如，缺失数据为字符串就替换为"n/a"，缺失数据为数值就替换为-1），可以传递一个字典，字典的值为要使用的字符串，字典的键为列名。

我们来看看美国的各州和人口普查局的地区组成的列表的导入过程（第36单元第2小节）。原始的CSV文件具有规则而稀疏的结构：

```
Northeast,New England,Connecticut
,,Maine
,,Massachusetts
,,New Hampshire
,,Rhode Island
,,Vermont
Northeast,Mid-Atlantic,New Jersey
,,New York
,,Pennsylvania
«more states»
```

它没有标题行，并且有许多空的单元格，不过我们知道如何填充列名和空单元格：

```
regions = pd.read_csv("code/regions.csv",
                      header=None,
```

```
                      names=("region", "division", "state"))
state2reg_series = regions.ffill().set_index("state")["region"]
state2reg_series.head()
```

```
⇛ state
⇛ Connecticut      Northeast
⇛ Maine            Northeast
⇛ Massachusetts    Northeast
⇛ New Hampshire    Northeast
⇛ Rhode Island     Northeast
⇛ Name: region, dtype: object
```

原先的state2reg是一个字典，不是一个series，不过pandas具有几乎适用于所有情况的转换函数：

```
state2reg = state2reg_series.to_dict()
```

```
⇛ {'Washington': 'West', 'South Dakota': 'Midwest', «more states»}
```

函数to_csv()将一个frame或一个series写到CSV文件中。

函数read_table()根据文件名或打开的文件句柄，从指定的表格文件中读取一个frame。本质上，它是一个使用制表符（而不是一个逗号）作为默认分隔符的read_csv()函数。

分块

如果你想从大型文件中分块读取表格数据，就需要进行分块。进行分块时，只需将参数chunksize（行数）传递给函数read_csv()即可。该函数没有实际地读取行，而是返回一个可以在for循环中使用的生成器。

6

让文件code/regions_clean.csv具有与code/regions.csv相同的数据，但没有遗漏的地区（region）和专区（division）。让我们假装这个文件**真的**很大，以至于不敢一下子读完这个文件。下面的代码创建一个用于迭代的TextFileReader对象和一个累加器series，每次从文件中读取5行。对于读取的每个片断，提取出"region"列，并对列中的值进行计数。然后将计数结果加到累加器。当某个键不存在于累加器中时，通过可选参数fill_value将其简单地设置为0，以避免出现nan。

```
chunker = pd.read_csv("code/regions_clean.csv", chunksize=5,
                      header=None, names=("region", "division", "state"))
accum = pd.Series()
for piece in chunker:
  counts = piece["region"].value_counts()
  accum = accum.add(counts, fill_value=0)
accum
```

```
⇛ Midwest      12
⇛ Northeast     9
⇛ South        17
⇛ West         13
⇛ dtype: float64
```

读取其他文件

函数`read_json()`尝试从JSON文件中读取一个frame。由于JSON文件不是一般的表格，它具有一定的层次结构，因此并非总是能将JSON数据强制转换为矩形格式。

函数`read_fwf()`从具有固定宽度数据的文件中读取一个frame。该函数使用的参数是`colspecs`（一行中列的开始位置和结束位置+1的若干个元组组成的列表）或`widths`（列宽组成的列表）。

函数`read_clipboard()`从系统剪贴板中读取文本，然后将其传递给`read_table()`。通过将数据复制到剪贴板，可以使用此函数一次性从Web页面中提取表格。

轮到你了

`pandas`的frame和series是非常便于使用的数据容器，它们在`numpy`数组之上增加了一个额外的抽象级别，并增强了一致性。frame和series非常适合将数据导入和导出到表格文件，结构化、重组、合并和聚合数据，以及对数据执行简单的乃至高级的算术运算。而且，不同于R语言的frame，`pandas`的frame的大小不受计算机内存的限制，限制它的只是你的想象力。

- **猞猁的捕获量***

  写一个程序，使用每年加拿大猞猁的捕获量[①]给出每十年的猞猁捕获总量，并将结果按逆序排列（最“有效益的”十年排在前面）。如果cache目录中不存在数据文件，则程序自动将数据下载到cache目录中。如果cache目录不存在，则程序将自动创建该目录。程序将计算结果保存到doc目录下的CSV文件中。如果doc目录不存在，则程序将自动创建该目录。

- **GDP与酒消费量****

  维基百科有很多关于人口统计学方面的数据，包括人均酒精消费量[②]和人均GDP[③]。编写一个使用这些数据给出GDP水平（高于平均水平、低于平均水平）与酒精消费水平（高于平均水平、低于平均水平）的交叉表。根据得出的表，分析这两个参数是否相关。

- **天气与酒精消费*****

  按州将酒精消费的历史数据与天气的历史数据相结合。使用交叉表来估计饮酒习惯是否与当地平均气温和总降雨量相关。换句话说，下雨的时候人们会喝更多酒吗？

  国家气候数据中心的网站[④]提供了天气的历史数据。

---

① vincentarelbundock.github.io/Rdatasets/csv/datasets/lynx.csv

② en.wikipedia.org/wiki/List_of_countries_by_alcohol_consumption_per_capita

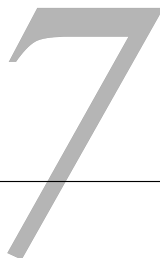
③ en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)_per_capita

④ www.ncdc.noaa.gov/cdo-web/

往下看去，目光扫过他全身的衣服，可以看到那些用绳扎在一起的洞，这在一定程度上确实类似于孩子眼中的网。

——英国小说家和短篇小说家Morley Roberts

第 7 章 使用网络数据



网络分析是数据分析的一个新兴领域。网络科学的网络理论和方法借鉴了数学（图论）、社会科学以及大量社会学内容。有些人还将网络分析称为“社交网络分析”，这一点我们（数据科学家）就不做评论了。

从数据科学的角度来看，网络是相互连接的对象构成的集合。实际上我们可以将任意类型的数字和非数字（文本）对象视为网络，只要对象之间存在一种相互连接的方式。结合相关的背景知识和研究领域，可以将网络对象称为“节点”“顶点”或“角色”，而将对象之间的连接称为“弧”“边”“链接”或“联系”。可以使用图形化的方式和数学的方式将网络表示为图形。

本章中，你将学习如何基于网络的和非网络的数据创建网络，了解网络度量并分析网络，尤其是如何计算和理解网络节点的中心性和社区结构。在本章中我们有两个好助手，分别是Anaconda的标准模块`networkx`和在学习社区检测[①]之前需要安装的`community`模块。

第 38 单元

概念剖析

在数学上，图是一组与边相连的节点的集合。边、节点和图的类型有许多种，它们具有不同的连接和解析方式。因此，在开始剖析图之前，我们先给出一些重要的定义。

① pypi.python.org/pypi/python-louvain/0.3

图的元素、类型和密度

如果图中至少有一条边是**有向的**（例如，一条节点A到节点B的有向边对应的是从A到B的连接，而不是从B到A的连接），则图本身就是有方向的，这样的图称为**有向图**。如果图中有平行边（即节点A可以通过多条边连接到节点B），则该图称为**多图**。从节点A到节点A的边称为**循环**。不存在循环和平行边的图称为**简单图**。

图的边可以分配权重。权重通常（但不是必须）是0和1之间的数字（包括0和1）。权重越大意味着节点之间的连接越强。具有加权边的图称为**加权图**。

边的权重是边的一种属性。边可能还有其他属性，包括数字、布尔和字符串类型的变量。图的节点同样可以具有类似的属性。

图的节点的度被定义为连接（入射）到节点的边数。对于有向图来说，度进一步区分为**入度**（以节点为终点的边数）和**出度**（以节点为起点的边数）。

图的密度d（$0 \leq d \leq 1$）表示图与**完全图**的接近程度。所谓完全图是指包含所有可能边的图。例如，一个具有e条边和n个节点的有向图，其密度为：

$$d = \frac{e}{n(n-1)}$$

相应的无向图的密度为：

$$d = \frac{2e}{n(n-1)}$$

图的结构

图以及图所引出的网络都是多样化、多面性、令人兴奋的对象。了解必要的术语有助于更好地掌握相关知识。

节点间的连通性是基于图上的游走过程得出的概念。一个**游走**过程是由一组边的序列构成的，其中一条边的端点是另一条边的起点。乘坐公共汽车从“家”节点到“地铁站A”节点，然后乘坐地铁从“地铁站A”到“地铁站B”，再从“地铁站B”步行到“我们的办公室”节点，我们就完成了一个图上的游走（虽然只有一部分是真正靠步行完成的）。我们称不与自身相交的游走（也就是说，游走过程中除了第一个和最后一个节点外，节点出现的次数都小于两次）为**路径**。一个闭合的路径称为**循环**。当我们结束艰难的一天回到家时，一天的游走就构成了循环①。

和测量现实生活中两个对象之间的距离一样，我们可以测量任意两个节点之间的距离，这个距离是连接节点的所有路径中最小的边数（有时也称为“跳”）。加权图没有距离的定义，不过有

① 注意不能原路返回，否则不构成路径，更不是循环。——译者注

时你可以假装图没有加权，仍然用边数来计算距离。对于有向图而言，从A到B的距离并不总是等于从B到A的距离。实际上，也许我们只能从A到B，而不能从B到A！从出生到死亡的生命历程就是一个伤感的却再清楚不过的例子。

图中任意两个节点之间的最大距离称为图的**直径**（D）。与圆不同，图没有面积。

连通分量或分量是图中这样一组节点的集合：集合中的每个节点都具有到达集合中的所有其他节点的路径。对于有向图来说，连通分量又分为强连通分量（有实际的连接路径）和弱连通分量（将边转换为无向边后才存在连接路径）。

如果图有多个分量，则最大的分量称为**巨型连通分量**（GCC）。GCC的规模通常很大，此时为了避免遇到连接不存在的问题，最好使用GCC，而不是整个图。

有时图的两个部分是互连的，但其连接是如此地微妙，以至于移除单条边后图就被分开了。这样的边被称为**桥**。

团是这样一组节点的集合：每个节点都与集合中的其他节点直接相连。D'Artagnan和三个火枪手就组成了一个团，包括任何奉行"人人为我，我为人人"原则的组合也一样。我们称图中最大的团为**最大团**。如果一个团不能通过向其中添加另一个节点而扩大，则称这个团为**极大团**。最大团一定是极大团，但反过来未必成立。完全图本身就是一个最大团。

星形图是这样一组节点的集合：集合中存在一个节点与其他所有节点相连接，但是其他节点之间不存在连接。星形图通常存在于分级的多层次系统中（例如公司、军事机构和互联网）。

直接与节点A相连接的一组节点称为节点A的**邻域**（A的G(A)）。邻域是**雪球效应**的关键部分。雪球效应是一种数据获取技术，通过随机选择的种子节点跟踪到其邻居节点，进一步到达邻居的邻居节点（二度邻域），并不断重复该过程。

节点A的局部集聚系数（或简称为集聚系数）是A的邻域所具有的边数CC(A)（不包括直接与A相连的边）除以最大可能的边数。换句话说，集聚系数是去除节点A后A的邻域的密度。星形图中任意节点的集聚系数为0。完全图中任意节点的集聚系数为1。可以把CC(A)看作G(A)的星形相似度或是其完全性的度量。

网络**社区**是这样一组节点的集合：集合中节点互连的边数远大于穿过社区边界的边数。**模块度**（m∈[−1/2, 1]）是社区结构质量的度量。它被定义为社区内节点的互连边数比例与随机情况下的边数比例之差。高模块化（m≈1）表征了一个拥有密集和清晰可见社区的网络。识别这样的社区可能是网络数据分析最重要的结果（具体内容参考第39单元）。

中心性

中心性是网络中节点重要性的度量。有多种类型的中心性度量，它们从不同方面对节点的重要性进行了衡量。为了方便起见，中心性通常被缩放到0（对应不重要的外围节点）和1（对应重

7

要的中心节点）之间。

- **度中心性**

  节点A的**度中心性**是A的邻居节点个数，也就是A的度或G(A)的大小。可以通过除以A的最大可能邻居数n-1，将其缩放到合适的范围。

- **接近中心性**

  节点A的**接近中心性**是其他所有节点到节点A的平均最短路径长度L_{BA}的倒数：

$$\mathrm{cc}_A = \frac{n-1}{\sum_{B \neq A} L_{BA}}$$

- **中介中间性**

  节点A的**中介中心性**是指网络中所有两个节点之间的最短路径中，经过A点的路径数量与最短路径总数量之比。

- **特征矢量中心性**

  节点A的**特征矢量中心性**被定义为A的所有邻居节点的特征矢量中心性的加权和，这是一个递归定义：

$$\mathrm{ec}_A = \frac{1}{\lambda} \sum_{B \in G(A)} \mathrm{ec}_B$$

最后两个中心性度量的计算代价非常大，而且对于大型网络可能并不实用。

第39单元

网络分析序列

掌握了适当的定义和公式后，我们接下来开启网络数据分析的宏伟篇章。

典型的网络分析序列包括以下步骤。

(1) 首先，识别离散实体以及实体间的关系。实体转化为网络节点，而实体间的关系转化为网络的边。如果关系是二元的（例如存在和不存在），就可以直接定义出网络的边。如果关系不是二元的，而是连续的或离散的，可以将它们看作加权的边，或者只将值等于或高于阈值的关系转换为未加权的边。后一种转换称为**抽样**。抽样阈值是从经验和实用的角度来选择的。如果阈值太高，则网络会分解成很多小的连通量，显得过于稀疏；如果阈值太低，网络就会失去社区结构，变得混乱。

(2) 计算各种网络度量：密度、分量的数目、GCC的大小、直径、中心性和集聚系数等。

(3) 识别网络社区。如果网络最终是模块化的，就可以给社区分配标签，将社区替换为“超节点”，并在导出的新网络上开展研究。

(4) 最后，和任何其他数据科学实验一样，都要对结果进行解释，并生成一个包含许多吸引眼球的图片的报告。

networkx模块几乎提供了你开展典型网络研究所需要的一切，只有一个例外：它生成的图片毫无吸引力，坦率地讲，图片的质量相当可悲。为了得到更佳的可视化效果，可以使用Gephi（请参阅第40单元第2小节）。

第 40 单元

使用 networkx

networkx模块包含构建、修改、探索、绘制、导出和导入网络的基本工具。它支持简单图、有向图和多图。你将学会如何通过添加和删除节点、边以及属性来构建和修改网络，如何计算各种网络度量（比如中心性），以及如何探索网络社区结构。

构建和修改网络

我们利用维基百科的数据①，根据已知的国际陆地边界信息（包括其长度信息）来构建一个关于国家的网络，并发掘该网络包含的信息。该网络图是无向的，不包含循环和平行边。

```
import networkx as nx

borders = nx.Graph()
not_borders1 = nx.DiGraph() # 仅作为参考
not_borders2 = nx.MultiGraph() # 仅作为参考
```

可以通过添加或删除单个节点（或单条边），甚至是一组节点（或一组边）来修改现有网络图。当某个节点被删除时，与其相连的所有边也随之删除。当某条边被添加时，边的端点对应的节点也被添加到图中，除非它们已经存在于图中。可以使用数字或字符串对节点进行标记：

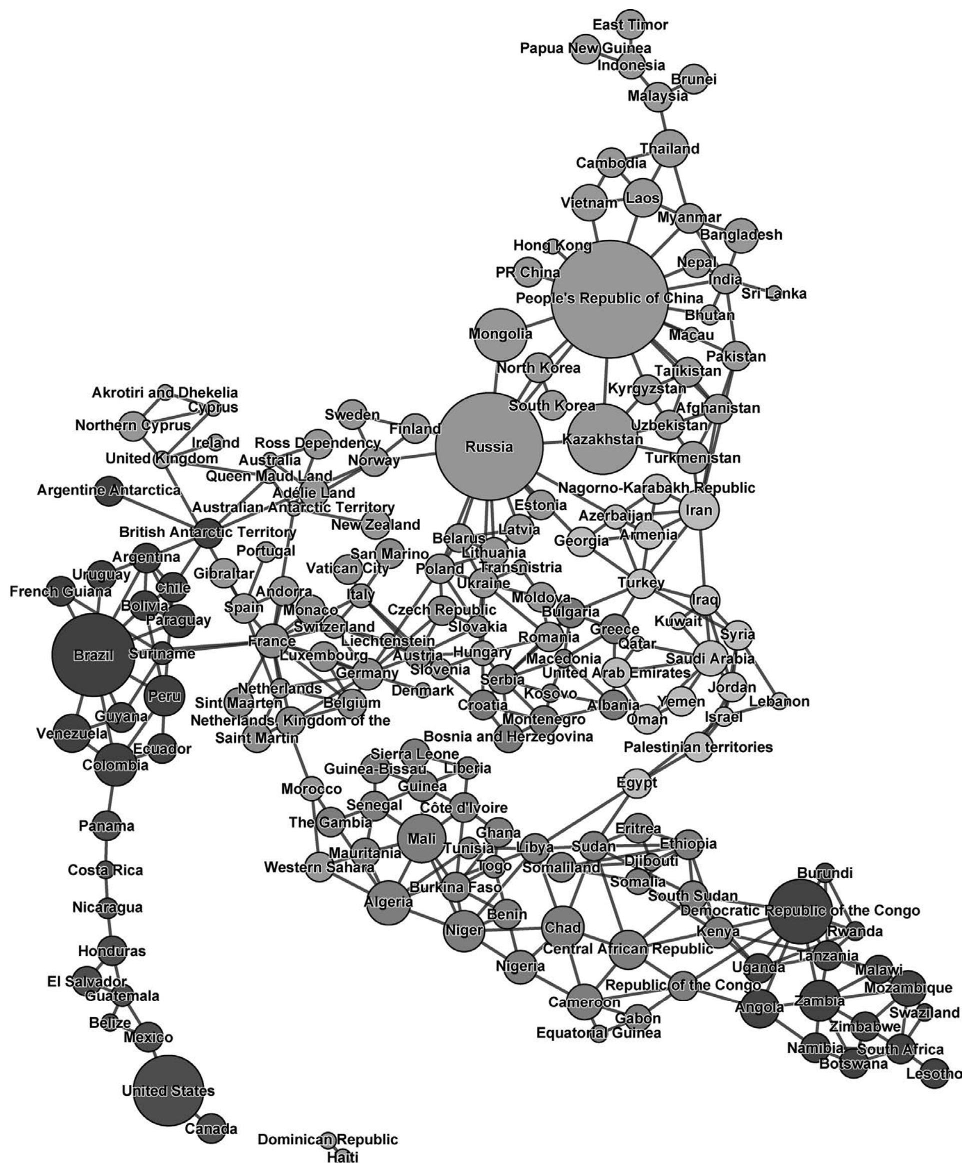
```
borders.add_node("Zimbabwe")
borders.add_nodes_from(["Lugandon", "Zambia", "Portugal", "Kuwait",
                        "Colombia"])
borders.remove_node("Lugandon")
borders.add_edge("Zambia", "Zimbabwe")
borders.add_edges_from([("Uganda", "Rwanda"), ("Uganda", "Kenya"),
```

① en.wikipedia.org/wiki/List_of_countries_and_territories_by_land_borders

7

```
            ("Uganda", "South Sudan"), ("Uganda", "Tanzania"),
            ("Uganda", "Democratic Republic of the Congo")])
```

当所有领土以及它们之间的连接添加完成后，你将看到一幅可读性很强的图，如下所示。



图中的节点大小表示其对应国家的陆地边界总长度，节点的不同颜色对应不同的网络社区（将在下一小节介绍）。

最后，可以使用`clear()`函数删除图中所有的节点和边，不过你不太可能会经常用到这个函数。

探索和分析网络

使用networkx进行网络分析和探索就如同调用几个函数和查看几个属性的值一样简单。下面使用len()函数返回图的“长度”（即图的节点数）。鉴于本书多次调用过Python的内置函数len()，相信你对于此处这样的用法应该不会感到惊讶。

```
len(borders)
```

```
⇒ 181
```

实际的节点列表可以通过nodes()函数、node属性和edge属性来获得（后一个属性也包含边的字典）。node和edge这两个属性都是只读的，不能通过修改它们来添加边或节点。（即使你尝试修改它们，networkx也不会保存你的更改。）边的列表也可通过函数edges()来获得：

```
borders.nodes()
```

```
⇒ ['Iran', 'Palestinian territories', 'Chad', 'Bulgaria', 'France', «...»]
```

```
borders.node
```

```
⇒ {{'Iran': {'L': 5440.0}, 'Palestinian territories': {},
⇒   'Chad': {'L': 5968.0}, 'Bulgaria': {'L': 1808.0}, «...»}
```

可以看到，字典给出了节点的属性（在我们的例子中，节点的属性是陆地边界的总长度）：

```
borders.edge
```

```
⇒ {'Iran': {'Nagorno-Karabakh Republic': {}, 'Turkey': {}, 'Pakistan': {},
⇒  'Afghanistan': {}, 'Iraq': {}, 'Turkmenistan': {}, 'Armenia': {},
⇒  'Azerbaijan': {}}, «...»}
```

edge属性是一个字典，每个节点都对应字典的一个条目。边的属性，比如“权重”，也包含在字典中。

```
borders.edges()[:5]
```

```
⇒ [('Iran', 'Nagorno-Karabakh Republic'), ('Iran', 'Turkey'),
⇒  ('Iran', 'Pakistan'), ('Iran', 'Afghanistan'), ('Iran', 'Iraq')]
```

可以通过neighbors()函数获取节点的邻居列表：

```
borders.neighbors("Germany")
```

```
⇒ ['Czech Republic', 'France', 'Netherlands, Kingdom of the', 'Denmark',
⇒  'Switzerland', 'Belgium', 'Netherlands', 'Luxembourg', 'Poland', 'Austria']
```

调用函数degree()、indegree()或outdegree()能算出节点邻域的长度，从而得出节点的度、入度和出度。当以无参数的方式调用这些函数时，它们返回由节点标签索引的度数字典。当使用节点标签作为参数调用它们时，它们返回该节点的度数。

```
borders.degree("Poland")
```

```
⇛ 7
```

```
borders.degree()
```

```
⇛ {'Iran': 8, 'Nigeria': 4, 'Chad': 6, 'Bulgaria': 5, 'France': 14,
⇛  'Lebanon': 2, 'Namibia': 4, «...»}
```

下面我们可以看一下哪个国家拥有最多的邻居：

```
degrees = pandas.DataFrame(list(borders.degree().items()),
                columns=("country", "degree")).set_index("country")
degrees.sort("degree").tail(4)
```

```
⇛ Country
⇛ Brazil                      11
⇛ Russia                      14
⇛ France                      14
⇛ People's Republic of China 17
```

一个大型网络的大型套件



使用真正的大型网络对于我们来说并非少见。(例如，Facebook的社交网络图有15.9亿个节点。初学者该如何使用这样的大型网络呢？)就像在纯Python中的实现一样，networkx并不是以高性能著称的。对于大型网络来说，应该使用NetworKit，它是一种高效的、可并行化的网络分析工具包。NetworKit的开发商声称："在一台16核服务器上，只需几分钟就能完成拥有30亿条边的网络图的社区检测。"[①]对于community模块而言，这是望尘莫及的。最重要的是，NetworKit模块可以与matplotlib、scipy、numpy、pandas和networkx相结合，这进一步增强了这个模块的吸引力。

有向图不存在集聚系数的定义，但必要时可以将有向图转换为无向图。clustering()函数返回包含所有给定节点的集聚系数构成的一个字典：

```
nx.clustering(not_borders1) # clustering()函数不能作用在有向图上!
nx.clustering(nx.Graph(not_borders1)) # 需要先转换为无向图!
nx.clustering(borders)
```

```
⇛ {'Iran': 0.2857142857142857, 'Nigeria': 0.5, 'Chad': 0.4, 'Bulgaria': 0.4,
⇛  'France': 0.12087912087912088, 'Lebanon': 1.0, 'Namibia': 0.5, «...» }
```

```
nx.clustering(borders, "Lithuania")
```

```
⇛ 0.8333333333333334
```

① networkit.iti.kit.edu

函数connected_components()、weakly_connected_components()和strong_connected_components()从图中返回各种连通分量（表示为节点的标签列表）的列表生成器。可以在迭代器表达式（for循环或列表解析）中使用生成器，或者使用内置的list()函数将其转换为列表。使用函数subgraph(G,n)可以得到由图G中节点列表n定义的子图。另外，也可以使用connected_component_subgraphs()的一系列函数等来计算连通分量，并将结果作为图的列表生成器：

```
list(nx.weakly_connected_components(borders)) # 此句代码不能运行!
list(nx.connected_components(borders)) # 此句代码可以运行!
```

⇒ **[{'Iran', 'Chad', 'Bulgaria', 'Latvia', 'France', 'Western Sahara', «...»}]**

```
[len(x) for x in nx.connected_component_subgraphs(borders)]
```

⇒ **[179, 2]**

Gephi它！



Gephi是"各种网络和复杂系统的交互式可视化探索平台"[1]，有人将它称为"网络分析的画笔"。虽然networkx具有自己的图形可视化支持（通过matplotlib的方式，详情参阅第41单元），但我更喜欢使用Gephi，因为它用途广泛，而且具有即时反馈的功能。

所有计算中心性的函数都返回一个以节点标签为索引的中心性字典或者单个节点的中心性。这些字典是用于构建pandas数据frame和具有索引的series的极佳组件。下面代码中的注释给出了不同中心性中具有最大值的国家：

```
nx.degree_centrality(borders) # 中国
nx.in_degree_centrality(borders)
nx.out_degree_centrality(borders)
nx.closeness_centrality(borders) # 法国
nx.betweenness_centrality(borders) # 法国
nx.eigenvector_centrality(borders) # 俄罗斯
```

管理属性

networkx使用字典实现图及其节点和边的属性。图具有节点的字典接口，节点具有其边的字典接口，边具有其属性的字典接口。你可以将属性名称和值作为可选参数传递给函数add_node()、add_nodes_from()、add_edge()和add_edges_from()：

```
# 边属性
borders["Germany"]["Poland"]["weight"] = 456.0
```

① gephi.org

```
# 节点属性
borders.node["Germany"]["area"] = 357168
borders.add_node("Penguinia", area=14000000)
```

使用可选参数data=True调用nodes()和edges()函数时，它们分别返回包含了所有属性的全部节点和边的列表：

```
borders.nodes(data=True)
```

```
⇒ [«...», ('Germany', {'area': 357168}), «...»]
```

```
borders.edges(data=True)
```

```
⇒ [('Uganda', 'Rwanda', {'weight': 169.0}),
⇒  ('Uganda', 'Kenya', {'weight': 933.0}),
⇒  ('Uganda', 'South Sudan', {'weight': 435.0}), «...»]
```

团和社区结构

函数find_cliques()和isolates()能检测出图中最大的团和孤立节点（即零度节点）。函数find_cliques()不能直接用于有向图（有向图应首先强制转换为无向图）。该函数返回一个节点列表的生成器，如下所示：

```
nx.find_cliques(not_borders1) # 不能直接用于有向图!
nx.find_cliques(nx.Graph(not_borders1)) # 转换为无向图才能正常运行
list(nx.find_cliques(borders))
```

```
⇒ [['Iran', 'Nagorno-Karabakh Republic', 'Armenia', 'Azerbaijan'],
⇒  ['Iran', 'Afghanistan', 'Pakistan'], «...»]
```

```
nx.isolates(borders)
```

```
⇒ ['Penguinia']
```

用于社区检测的community模块并不包含在Anaconda发行版中，必须单独安装，而且该模块不支持有向图。

函数best_partition()使用Louvain方法并返回社区的划分结果，划分结果是一个以节点标签作为索引的字典，并用不同的数字序号区分不同的社区。函数modularity()给出社区的模块度：

```
import community
partition = community.best_partition(borders)
```

```
⇒ {'Uganda': 0, 'Zambia': 1, 'Portugal': 2, 'Bulgaria': 2, «...»}
```

```
community.modularity(partition, borders)
```

```
⇒ 0.7462013020754683
```

如果模块度太低（远小于0.5），网络就不存在清晰的社区结构，至少你不能依赖给出的划分结果。

输入和输出

networkx具有一系列从文件读取网络数据并将数据写入文件的读写函数,我们只需要负责打开和关闭文件（必要时负责创建文件）。某些函数要求文件以二进制模式打开。通过下表可以了解其中的部分函数。

表4　networkx的一些输入和输出函数

类　型	读	写	文件后缀
邻接列表	`read_adjlist(f)`	`write_adjlist(G, f)`	无特定后缀
Edge list	`read_edgelist(f)`	`write_edgelist(G, f)`	无特定后缀
GML	`read_gml(f)`	`write_gml(G, f)`	.gml
GraphML	`read_graphml(f)`	`write_graphml(G, f)`	.graphml
Pajek	`read_pajek(f)`	`write_pajek(G, f)`	.net

```
with open("borders.graphml", "wb") as netfile:
  nx.write_pajek(borders, netfile)
with open("file.net", "rb") as netfile:
  borders = nx.read_pajek(netfile)
```

不是任意格式的文件都能支持所有网络相关的特性。你可以在Gephi网站[1]上查看关于不同文件格式和网络特性的更多信息，了解这些内容有助于为你的图选择正确的输出格式！

轮到你了

网络分析是一项极具感染性的工作。学会网络分析后，随处都可以找到网络分析的影子，甚至是在莎士比亚的作品中。[2]网络、中心性和社区方面的思维方式给你增加了另一个层次的数据分析技能，因此不要犹豫，开始下面的强化练习吧。

- **中心性的相关问题***

  从斯坦福大网络数据集[3]（由J. Leskovec和A. Krevl编写）中下载Epinions.com网站用户的社交网络图，并提取出规模排名第十的社区。编写一个程序，计算并显示本章中提到的所有网络中心性度量之间的两两相关性（加入集聚系数可以使问题变得更加有趣）。建议你使用pandas的frame来存储所有中心性。你可能需要阅读第47单元第2小节来了解如何

① gephi.org/users/supported-graph-formats/
② www.slideshare.net/DmitryZinoviev/desdemona-52994413
③ snap.stanford.edu/data/soc-Epinions1.html

7

使用pandas计算相关性。

是否存在强相关的一对中心性？

- 莎士比亚作品**

  可以从麻省理工学院[1]获得完整的威廉·莎士比亚作品。编写一个程序，创建出莎士比亚的所有戏剧以及100个最常用的非停止词组成的网络。如果词干出现在戏剧中，词干节点就和戏剧节点相连，边的权重等于该词干的使用频率。接下来用程序识别网络社区，并使用community模块给出社区模块度和成员节点。

  根据你对莎士比亚的了解，分析程序给出的社区划分是否有意义？

- 边界网络***

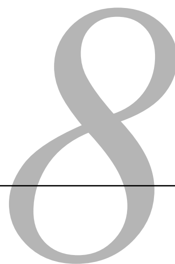
  使用维基百科数据、networkx和Gephi重新建立国家和边界的网络，要求包含节点大小和社区的信息（给节点分配相应的属性，这些属性可以在Gephi中用于设置节点的大小和颜色）。

① shakespeare.mit.edu

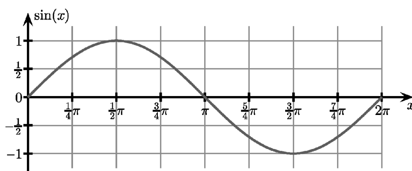
“一方面，我在为自己策划（plot[①]）；另一方面，对于别人的策划，我将计就计（counterplot）。”特雷瑟姆神秘地回答。

——英国历史小说家William Harrison Ainsworth

第8章 绘图



绘制数据是任何探索性或预测性数据分析的重要组成部分，甚至可能是报告撰写中最重要的部分。坦率地说，不会有人愿意看没有图片的报告，哪怕是毫不相关的图，如同这幅优美的正弦波：



可编程的绘图方法主要有三种。我们从一张空白画布开始进行**增量**绘图，然后使用专门的函数逐步添加图形、坐标轴、标签和图例等。最后，显示出所绘制的图像并将其保存到文件中（保存操作是可选的）。增量绘图工具的例子包括R语言函数`plot()`、Python的`pyplot`模块以及`gnuplot`的命令行绘图程序。

集成绘图系统将描述图形、图表、坐标轴、标签和图例等的所有必要参数传递给绘图函数。我们可以实现即时绘制、装饰并保存最终绘图。R语言的`xyplot()`函数是集成绘图工具的一个例子。

最后，**图层**工具用于展示要绘制的内容、如何绘制以及可展示其他任意特征的虚拟“图层”。我们可以根据需要给“plot”对象添加更多图层。R语言函数`ggplot()`是分层绘图工具的一个例子。（为了兼顾美学，Python的`matplotlib`模块提供了`ggplot`的绘图风格。）

本章中，你将看到如何使用`pyplot`进行增量绘图。

① plot既有“绘图”之意，又有“策划”之意。——编者注

第 41 单元

使用 PyPlot 进行基本绘图

matplotLib模块（更明确地，也就是其子模块pyplot）提供了针对numpy和pandas的绘图。

通过调用NIAAA的酒精监控报告来开始我们的试验。在第31单元我们已经将该报告转换为frame，接下来我们将绘制各个州每种酒的消费量。可惜的是，增量绘图系统没有一个能完成所有绘图的函数，让我们来看一个完整的例子：

pyplot-images.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# 在此之前NIAAA frame已完成了pickle操作
alco = pickle.load(open("alco.pickle", "rb"))
del alco["Total"]
columns, years = alco.unstack().columns.levels

# 直接从文件中读出州的缩略语
states = pd.read_csv(
    "states.csv",
    names=("State", "Standard", "Postal", "Capital"))
states.set_index("State", inplace=True)

# 将酒的销量按2009年的统计数据排序
frames = [pd.merge(alco[column].unstack(), states,
                   left_index=True, right_index=True).sort_values(2009)
          for column in columns]

# 有多少年的数据?
span = max(years) - min(years) + 1
```

代码首先导入了所有必要的模块和frame。然后将NIAAA数据和州名的缩写组合成一个frame，并通过酒的种类将其分成三个单独的frame。下面的代码片段负责绘图。

pyplot-images.py

```
# 选择一种优美的绘图样式
matplotlib.style.use("ggplot")

STEP = 5
# 一个frame绘制在一幅子图中
for pos, (draw, style, column, frame) in enumerate(zip(
❶        (plt.contourf, plt.contour, plt.imshow),
        (plt.cm.autumn, plt.cm.cool, plt.cm.spring),
        columns, frames)):
```

```
    # 选中第2行第2列的子图
❷   plt.subplot(2, 2, pos + 1)

    # 绘制frame
❸   draw(frame[frame.columns[:span]], cmap=style, aspect="auto")


    # 加入图饰
❹   plt.colorbar()
    plt.title(column)
    plt.xlabel("Year")
    plt.xticks(range(0, span, STEP), frame.columns[:span:STEP])
    plt.yticks(range(0, frame.shape[0], STEP), frame.Postal[::STEP])
    plt.xticks(rotation=-17)
```

函数`imshow()`、`contour()`和`contourf()`（标记为❶处）分别将矩阵显示为图像、等高线图和填充等高线图。不要在同一个子图中使用这三个函数（或任何其他绘图函数），因为它们会在先前绘制的图上叠加新的绘图，除非这正是你想要的。可选参数`cmap`（标记为❸处）为绘图指定了一个预制调色板（彩色图）。

可以将相同或不同类型的几个子图包装成一个主图（标记为❷处）。函数`subplot(n, m, number)`将主图划分为n个虚拟行和m个虚拟列，并用`number`选择子图编号。子图从第1列开始先逐列编号，再逐行编号。（左上角的子图为1，右侧的下一子图为2，以此类推。）所有绘图命令只影响最新选中的子图。

请注意，图像的原点在左上角，Y轴是指向下的（这是计算机图形学中的绘图方式），但其他所有图形的原点均位于左下角，而Y轴是指向上的（这是数学中的绘图方式）。另外，具有相同数据的图像和等高线图在默认情况下具有不同的纵横比，不过你可以通过传递`aspect="auto"`选项使它们看起来比较接近。

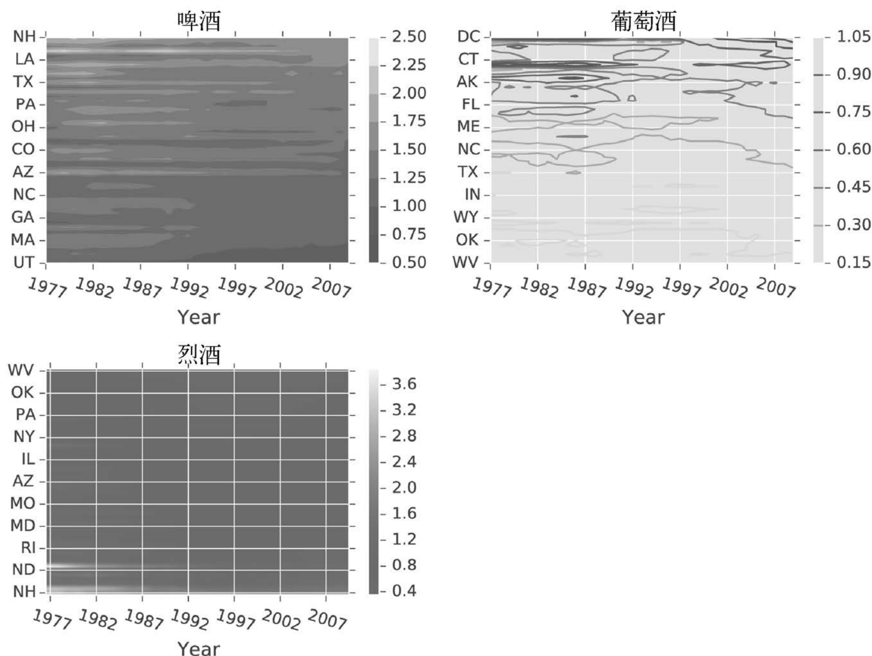
函数`colorbar()`、`title()`、`xlabel()`、`ylabel()`、`grid()`、`xticks()`、`yticks()`和`tick_params()`（标记为❹处）可以将相应的装饰添加到绘图中。（我们将在第43单元中重新考察这些函数。）`grid()`函数实际上实现了网格的打开和关闭，所以图形是否具有网格取决于你起初是否使用了`grid()`函数，而这反过来又受到绘图风格的控制。

函数`tight_layout()`可以调整子图，使它们看起来美观和紧凑。让我们来看看绘图的结果：

pyplot-images.py

```
plt.tight_layout()
plt.savefig("../images/pyplot-all.pdf")
#plt.show()
```

8



函数`savefig()`将当前图形保存到文件中。该函数的第一个参数是文件名或已打开文件的句柄。如果传递的是文件名，`savefig()`会尝试通过文件的扩展名猜测图像格式。该函数支持多种流行的图像文件格式，但GIF除外。

函数`show()`将绘图显示在屏幕上。它同时也清空了画布，但如果你只想清空画布，那就单独调用`clf()`函数。

第 42 单元

了解其他绘图类型

除了等高线和图像，`pyplot`还支持多种更为传统的绘图类型：条形图、箱形图、直方图、饼图、线图、对数和双对数图、散点图、极坐标图、步阶图，等等。在线的`pyplot`图库[①]提供了许多示例，下表列出了`pyplot`的许多绘图函数。

① matplotlib.org/gallery.html

表5　pyplot的一些绘图类型

绘图类型	函　数
竖直条形图	bar()
水平条形图	barh()
“有须”的箱形图	boxplot()
误差条形图	errorbar()
直方图（垂直或水平）	hist()
双对数图	loglog()
X轴对数图	semilogx()
Y轴对数图	semilogy()
饼图	pie()
线图	plot()
日期图	plot_dates()
极坐标图	polar()
散点图（点的大小和颜色可以控制）	scatter()
步阶图	step()

第 43 单元

精通绘图装饰

你可以用pyplot实现对绘图全方面的控制。

可以使用函数xscale(scale)和yscale(scale)来设置和更改轴的刻度（"linear"和"log"），可以使用函数xlim(xmin, xmax)和yimim(ymin, ymax)来设置和更改轴的上下限。

可以设置和更改字体、图形和背景颜色，以及字体和点的大小和样式。

还可以使用annotate()添加注释，用arrow()添加箭头，用legend()添加图例。通常来说，要了解完整的装饰函数及其参数列表，需要参阅pyplot文档，但是目前你至少可以将一些箭头、注释和图例添加到你已经熟悉的NIAAA图中：

8

pyplot-legend.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# 在此之前NIAAA frame已完成了pickle操作
alco = pickle.load(open("alco.pickle", "rb"))

# 选择合适的数据
BEVERAGE = "Beer"
years = alco.index.levels[1]
states = ("New Hampshire", "Colorado", "Utah")
```

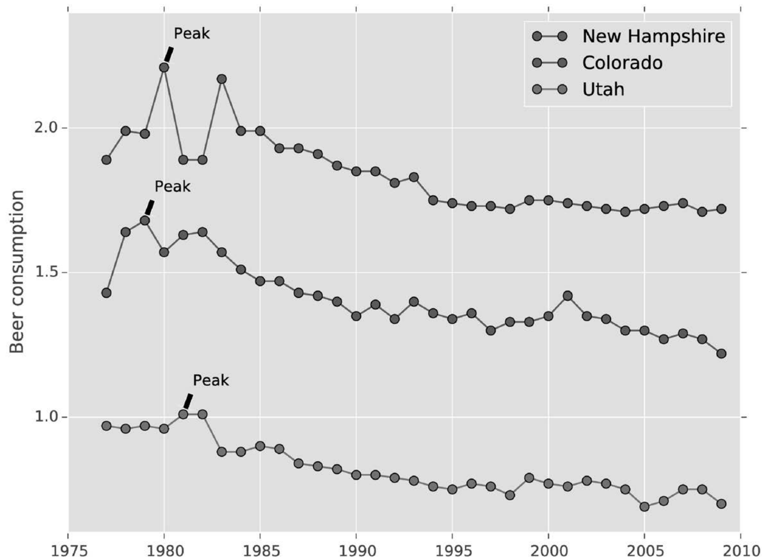
```
# 选择一个优美的绘图样式
plt.xkcd()
matplotlib.style.use("ggplot")

# 绘制图表
for state in states:
    ydata = alco.ix[state][BEVERAGE]
    plt.plot(years, ydata, "-o")
    # 添加带箭头的注释
    plt.annotate(s="Peak", xy=(ydata.argmax(), ydata.max()),
                 xytext=(ydata.argmax() + 0.5, ydata.max() + 0.1),
                 arrowprops={"facecolor": "black", "shrink": 0.2})

# 添加标签和图例
plt.ylabel(BEVERAGE + " consumption")
plt.title("And now in xkcd...")
plt.legend(states)

plt.savefig("../images/pyplot-legend-xkcd.pdf")
```

这里显示的三线图表明了三个州的啤酒消费动态（实际上，这三个州就是啤酒消费量最多、中等和最低的州）。

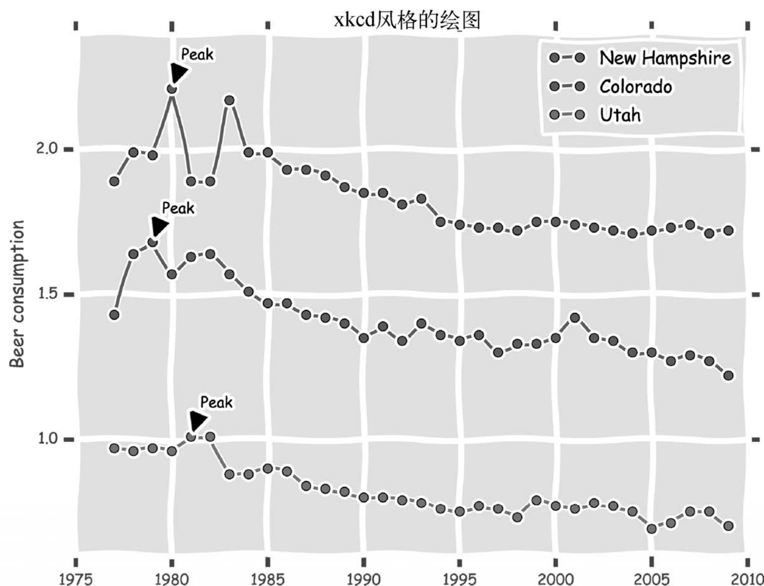


关于Unicode的说明



如果你的绘图包含Unicode字符（即非拉丁字符），那么你可能需要在绘制文本之前更改默认字体，具体的方法是将函数调用用`matplotlib.rc("font", family="Arial")`添加到绘图脚本的第一行。

最后，可以使用函数`xkcd()`将图案的风格改为流行的xkcd[①] Web漫画风格。（该函数仅对在调用它之后添加的绘图元素产生影响。）由于某些原因，我们无法将图形保存为PostScript文件，但其他格式的文件都是正常的。尽管如此，你应该避免在官方的演示中包含xkcd样式的绘图，因为它们看起来好像是出自醉汉之手（参见下图）。不管怎样，这个演示本身就是关于酒精消费量的。



`pyplot`模块本身已经非常强大了，与`pandas`相结合后还会更加出色，下面我们就来见识一下。

第 44 单元

用 pandas 绘图

`pandas`的frame和series都支持`pyplot`绘图。使用无参数的方式调用`plot()`函数时，可以绘制series或所有frame列的具有标签的线图。如果调用`plot()`时指定了可选参数`x`和`y`，则函数将相应地绘制列`x`与列`y`。

通过调整可选参数`kind`，`pandas`还可以支持其他类型的绘图。参数的允许值如下：绘制条形图的`"bar"`和`"barh"`、直方图的`"hist"`、箱形图的`"box"`、密度图的`"kde"`、面积图的`"area"`、散点图的`"scatter"`、六边形图形的`"hexbin"`，以及饼图的`"pie"`。所有的绘图都可以进行各种装饰，例如图例、色彩条、可控点的尺寸（选项`s`）和颜色（选项`c`）。

① xkcd.com

举个例子，让我们绘制整个NIAAA观察期间新罕布什尔州的葡萄酒与啤酒的消费量，并根据年份对每个数据点进行着色：

scatter-plot.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# 在此之前NIAAA frame已完成了pickle操作
alco = pickle.load(open("alco.pickle", "rb"))

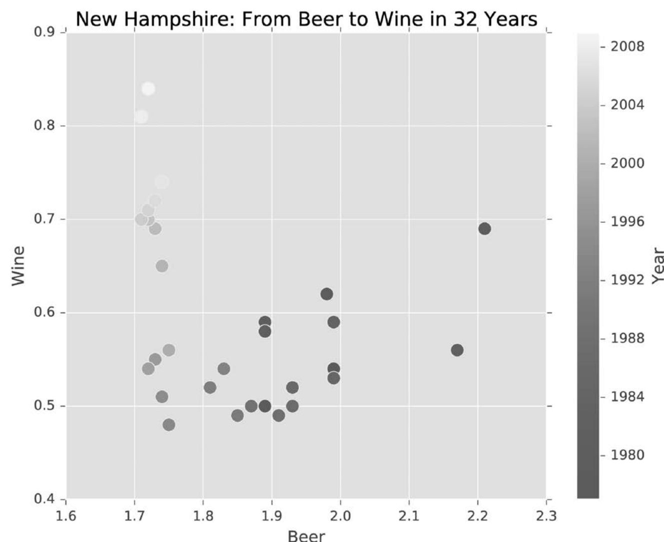
# 选择一个优美的绘图样式
matplotlib.style.use("ggplot")

# 绘制散点图
STATE = "New Hampshire"
statedata = alco.ix[STATE].reset_index()
statedata.plot.scatter("Beer", "Wine", c="Year", s=100, cmap=plt.cm.autumn)

plt.title("%s: From Beer to Wine in 32 Years" % STATE)
plt.savefig("../images/scatter-plot.pdf")
```

从绘制的图中不难看出，在32年的观察期中，新罕布什尔州经历了从喝啤酒到喝葡萄酒的转变。但愿这样的过程并不痛苦，也没有造成人口减少。

我们通过子模块`pandas.tools.plotting`来结束对绘图部分的介绍。该模块具有用于绘制安德鲁斯曲线`andrews_curves()`、滞后图`lag_plot()`和自相关`autocorrelation_plot()`的工具。但更重要的是，`pandas.tools.plotting`具有用于散点矩阵的工具。**散点矩阵**是一个非常优秀的探索数据的工具，它对应的函数实现是`scatter_matrix()`，该函数能显示主对角线中每列数据的直方图以及所有其他矩阵中任意两列的双变量散点图。



scatter-matrix.py

```
from pandas.tools.plotting import scatter_matrix
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

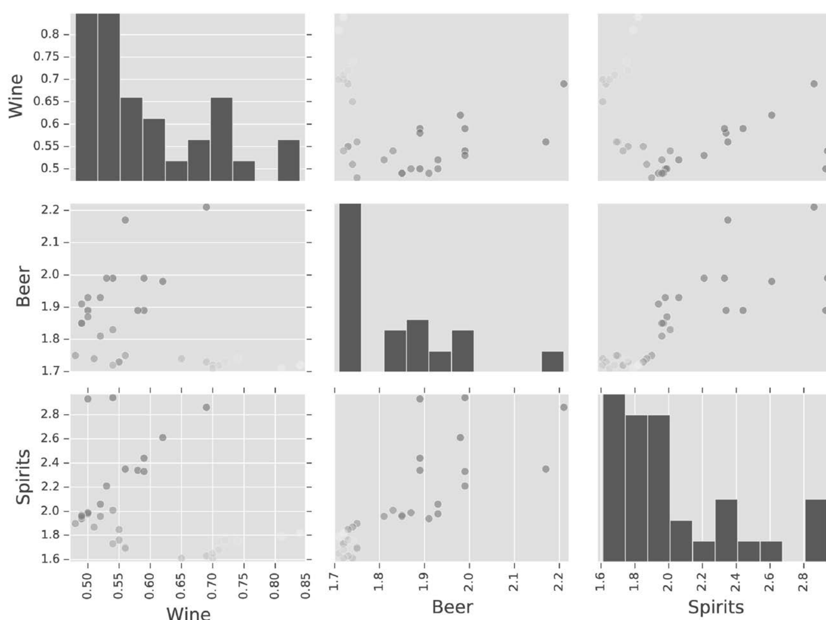
# 在此之前NIAAA frame已完成了pickle操作
alco = pickle.load(open("alco.pickle", "rb"))

# 选择一个优美的绘图样式
matplotlib.style.use("ggplot")

# 绘制散点矩阵
STATE = "New Hampshire"
statedata = alco.ix[STATE].reset_index()
scatter_matrix(statedata[["Wine", "Beer", "Spirits"]],
               s=120, c=statedata["Year"], cmap=plt.cm.autumn)

plt.tight_layout()
plt.savefig("../images/scatter-matrix.pdf")
```

我们再次对新罕布什尔州当地人的饮酒习惯进行分析。这一次，我们将32年记录中每年三种酒的六对两两组合同时绘制在一幅图中：



轮到你了

数据的绘制（以及更一般的数据可视化）并非徒劳的练习。不必说一图胜千言了，在数据科

学中，一幅图可能表示数百万甚至数十亿的数值观测。数据可视化为你提供了强大的工具，既可用于探索性的工作（给自己看），也可用于数据的展示（给同事和数据赞助者看）。

- ❑ **美国派***

  编写一个程序，将美国的所有州按首字母分组，并用一个饼图显示分组结果或保存为PDF文件。为了完成该项目，你需要用到州的名称或其缩写的列表，这个列表可以从命名网站获取[①]。

- ❑ **加州的人口****

  编写一个程序，使用美国人口普查局的数据[②]来显示2001年至2009年间加州人口（相对于美国总人口）的变化规律。

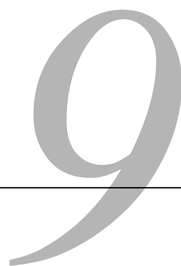
① www.stateabbreviations.us

② www.census.gov/popest/data/historical/2000s/vintage_2009/state.html

我不得不将这些大量的统计数据缩减为少量显著的事实。对我来说，统计数据和图片实在是太多了。

——美国作家、幽默大师马克·吐温

第 9 章 概率与统计



概率论和统计学的研究对象是随机现象，主要是随机抽样形式的随机现象，比如随机数和随机分类变量。

概率论关注随机样本的来源和产生。我们通过适当的概率分布得到随机样本，并将它们用于：

- 模拟合成原始数据，并用于模型测试（就像我们在第30单元中所做的）
- 将原始数据分解成测试集和训练集（参见第48单元）
- 支持随机机器学习算法（比如第51单元中的随机决策森林）

另一方面，统计学则主要研究已收集的随机样本的属性。实验的原始数据几乎总是具有不确定性和不可预测性。我们将使用各种统计度量来描述因变量的观测值以及因变量和自变量之间的相互作用。

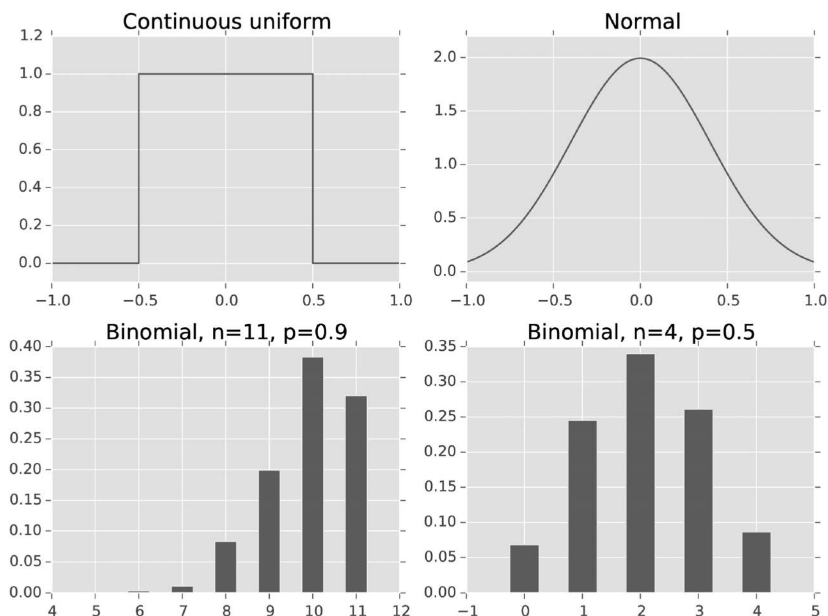
概率论和统计学都是具有丰富内涵和外延的数学领域。要学习它们的话，不能只阅读一本参考书中的一个章节。事实上，你可能已经对概率论和统计学有了一定的了解。本书的这部分内容只是对关键概念的简单回顾和总结。我们首先回顾一些概率理论，进而给出各种统计度量的数学定义，最后用Python的方式计算它们。

第 45 单元 回顾概率分布

概率分布为每个可能的随机数（离散分布）或随机数的取值范围（连续分布）分配一个概率。换句话说，它告诉我们，与其他事件相比，某些事件是否更有可能出现。常见的概率分布包括均

匀分布（连续和离散）、正态分布（连续）和二项分布（离散）。

可以通过概率质量函数（PMF，用于离散分布）或概率密度函数（PDF，用于连续分布）来指定概率分布。PDF描述了随机变量取得某个给定值的相对可能性，PMF给出了离散随机变量等于某个值的概率。下图给出了一些常见的概率分布的PDF（顶上一行）和PMF（底下一行）。（无疑，这些图是用`pyplot`绘制的，请参考第41单元）。



我们来简单看一下几种最有趣的概率分布类型。

均匀分布

均匀分布的所有结果具有相同的可能性。对于一个数值类型的随机变量来说，如果只有其取值范围是已知的，我们通常就使用均匀分布来表示该随机变量。

正态分布

正态分布也称为高斯分布或钟形曲线。如果一个实数随机变量的分布本身未知，但分布的平均值和标准差是已知的，就可以用正态分布来表示该随机变量。

二项分布

二项分布是$n > 0$个独立二元试验（比如“投掷硬币”）序列中成功次数的概率分布，其中单

次实验成功（掷出“头像”）的概率为$0 \leqslant p \leqslant 1$。成功次数的期望值为$n \times p$。当$n = 1$时，二项分布变为**伯努利分布**。

如果成功概率p为0.5，且独立重复试验次数为无限次，对应的二项分布随机变量实际上就是一个正态分布变量。换句话说，正态分布是无限次等概率二元试验的累积结果。

长尾



一些概率分布（例如帕累托分布，又称为Zipf分布或幂律分布）具有一条“长尾”，即分布的PDF无限向右或向左延伸，类似一个身体隆起并带有长尾的生物。长尾意味着远离均值的样本依然具有非零的概率，这样的样本可能会出现在数据中。

实际的原始数据可能恰好与某个理论分布相吻合，也有可能不存在任何适用于该数据的理论分布。但即便不存在吻合的分布，仍然可以通过计算各种统计度量来得出重要的结论，具体内容将在下一个单元介绍。

第 46 单元

回顾统计度量

从探索性（非基于推理的）数据科学的角度来看，统计学回答了四个重要问题。

- **数据位于何处？**

  样本均值是所有观察值的平均值：

$$\bar{x} = \sum x_i / N$$

  当数据接近正态分布（也即“钟形”分布）且标准差较小时，可以使用样本均值来表示整个样本。

- **数据分布有多广？**

  样本标准差是数据分散程度的度量，其计算方式为数据与样本均值的均方差的平方根：

$$s_x = \sqrt{\frac{\sum (x_i - \bar{x})^2}{N-1}}$$

  s_x越大，数据分布得越广。

- ❑ 数据分布的偏斜程度有多大？

  样本偏斜度是概率分布不对称性的度量。零偏斜度意味着分布是对称的。对于单峰分布（具有一种模式的分布）来说，负的偏斜度表示概率密度函数左侧的尾部长于右侧。

- ❑ 两个变量是否相关？

  样本协方差是衡量两个随机变量（的变化）接近程度的度量。X与自身的协方差称为方差，记为s^2（标准差的平方）。

$$q_{xy} = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{N - 1}$$

  皮尔逊（Pearson）相关系数，又称为相关系数或相关，是协方差的归一化：

$$r_{xy} = \frac{\sum (x_i y_i - N\overline{xy})}{(N-1)s_x s_y}$$

  相关性的取值范围是[−1…1]，请参考下表。相关性的取值较大意味着变量是相关的，取值较低意味着变量是反相关的。零相关意味着变量不是线性相关的。

表6　两个变量之间的线性关系的类型

	$r \ll 0$	$r=0$	$r \gg 0$
$p \leqslant 0.01$	强反相关	不线性相关	强相关
$p > 0.01$		不线性相关	

强相关性并不意味着变量存在因果关系。两个变量呈高度相关或反相关，其可能的原因是它们都由相同因素导致（混杂变量），但也可能只是巧合。例如，当白天变长黑夜变短，就会出现更多溺水身亡的人，但是溺水的原因并不是白天变长，而是因为夏天的到来既使得白天变长，也同时促使游泳的人增多！

类似地，两个变量不线性相关并不意味着二者就没有关系：它们之间可能是非线性的关系。当没有观察到明显的线性相关性时，不要失望。应该再分析一下其他的关系模型，比如第50单元中提到的聚类。

人口和样本



虽然你可能对统计推断（从样本数据中推测出人口可能具有的特征的艺术）不感兴趣，但仍需要明白，大多数时间数据科学家处理的并不是整个观测人群，而是量比较小的样本。本单元列出的所有统计度量都不是真值——它们是样本**估计值**。

当样本的观测次数较少时，两个变量之间的相关性可能很大，但相关性不一定显著。显著性的度量被称为***p*值**。*p*的值越小越好，不过$p \leqslant 0.01$就可以认为是足够好了。

现在我们可以转到Python了。我们将学习如何从分布中得出样本，并以Python的方式计算各种统计度量。

第 47 单元

以 Python 的方式完成统计

Python对随机数和统计的支持分散在以下多个模块中：`statistics`、`numpy.random`、`pandas`和`scipy.stats`。

生成随机数

`numpy.random`模块中包含所有主要概率分布的随机数生成器。

本书在第1章中就指出，数据分析的代码应该是可复现的：任何人都应该能够使用相同的输入数据运行相同的程序，并获得相同的结果。因此，你应该始终使用`seed()`函数来初始化伪随机种子，否则，随机数生成器在每个程序运行时会产生不同的伪随机序列，这可能使结果难以乃至不可能复现。

```
import numpy.random as rnd
rnd.seed(z)
```

下面的函数产生均匀、正态以及二项分布的整数和实值随机数。这些函数都返回一个具有`shape`或`size`的`numpy`数组(`shape`是一个由各个维度组成的列表),除非`size`的参数设置为`None`。

```
rnd.uniform(low=0.0, high=1.0, size=None)
rnd.rand(shape) # 等同于uniform(0.0, 1.0, None)
rnd.randint(low, high=None, size=None)
rnd.normal(loc=0.0, scale=1.0, size=None)
rnd.randn(shape) # 等同于normal(0.0, 1.0, shape)
rnd.binomial(n, p, size=None)
```

当需要设置预测实验并将数据分为训练集和测试集时，二项分布是必不可少的(更多关于预测实验的内容，请参考第48单元)。令训练集的相对大小为*p*，测试集的相对大小为1–*p*。你可以准备一个二值序列，然后将其转换为`True`和`False`值的布尔序列，最后通过`pandas`的frame选出两个集合：

```
selection = rnd.binomial(1, p, size=len(data)).astype(bool)
training = df[selection]
testing = df[-selection]
```

计算统计度量

作为一种快速而随性的计算统计度量的方式，statistics模块（本书首次提及该模块是在第14单元）提供低端函数mean()和stdev()。

pandas的frame和series具有用于计算与其他frame和series的相关性和协方差的函数，也有计算frame的列之间的两两相关性和协方差（但没有p值）以及其他统计度量的函数。

让我们再次使用第31单元中被转换为frame的NIAAA监控报告，来探索pandas的统计功能。（我们不是醉汉！我们是数据科学家：一旦得到了一个好的数据集，我们将最大限度地使用它！）准备两个series，然后计算它们的相关性、协方差和偏斜度：

```
beer_seriesNY = alco.ix['New York']['Beer']
beer_seriesCA = alco.ix['California']['Beer']

beer_seriesNY.corr(beer_seriesCA)

⇒ 0.97097785391654789

beer_seriesCA.cov(beer_seriesNY)

⇒ 0.017438162878787872

[x.skew() for x in (beer_seriesCA, beer_seriesNY)]

⇒  [0.16457291293004678, 0.32838100586347024]
```

我们也可以对frame应用相同的函数：

```
frameNY = alco.ix['New York']

frameNY.skew()

⇒ Beer       0.328381
⇒ Wine       0.127308
⇒ Spirits    0.656699
⇒ dtype: float64

frameNY.corr() # 所有的两两相关性

⇒              Beer      Wine   Spirits
⇒ Beer     1.000000  0.470690  0.908969
⇒ Wine     0.470690  1.000000  0.611923
⇒ Spirits  0.908969  0.611923  1.000000

frameNY.cov() # 所有的两两协方差

⇒              Beer      Wine   Spirits
⇒ Beer     0.016103  0.002872  0.026020
⇒ Wine     0.002872  0.002312  0.006638
⇒ Spirits  0.026020  0.006638  0.050888
```

后两个函数分别返回包含所有两两相关性及所有两两协方差的frame。

我们还可以将一个series和一个frame相关联，也可以将一个frame和另一个frame相关联。例如，使用美国人口普查局2000年至2009年间的数据[1]，分析纽约的酒精消费与国家人口之间是否存在某种相关性：

```
# 移除最后两行：因为这两行数据包含了对未来的估计值
pop_seriesNY = pop.ix["New York"][:-2]
# 将索引由日期变为整数年
pop_seriesNY.index = pop_seriesNY.index.str.split().str[-1].astype(int)

frameNY.ix[2000:2009].corrwith(pop_seriesNY)

⇛ Beer      -0.520878
⇛ Wine       0.936026
⇛ Spirits    0.957697
⇛ dtype: float64
```

请注意，frame和series中的行是以相反的顺序排列的。pandas非常智能，可以使用行索引来匹配正确的行。当然，真正起作用的是数据对齐机制。（第36单元详细讨论了数据对齐方式。）

要估计相关性的显著程度，请使用scipy.stats模块中的pearsonr()函数。该函数返回相关性和p值，但它不能与pandas的frame集成，也不支持索引，所以你需要将索引对齐并将结果转换回frame。

```
from scipy.stats import pearsonr
# 手动排列索引
pop_sorted = pop_seriesNY.sort_index()
alco_10 = alco.ix['New York'][-10:]
# 使用列表解析的方式计算所有的相关性和p值
corrs = [(bev,) + pearsonr(alco_10[bev], pop_sorted)
         for bev in alco_10.columns]
# 将列表转换为frame
pd.DataFrame(corrs, columns=("bev", "r", "p-value")).set_index("bev")

⇛                r  p-value
⇛ bev
⇛ Beer   -0.520878  0.122646
⇛ Wine    0.936026  0.000068
⇛ Spirits 0.957697  0.000013
```

请注意，“啤酒”相关性的p值是非常高的。根据第46单元中的表6，我们得出结论，人口与啤酒消费之间不存在线性关系。

我们现在完全有能力重新审视第36单元中的交叉表的例子。根据交叉表的分析，我们得出2009年的人均啤酒消费量和葡萄酒消费量可能是线性独立的。皮尔逊相关性分析完全证实了我们的说法：

[1] www.census.gov/popest/data/historical/2000s/vintage_2009/state.html

```
alco2009.corr()
```

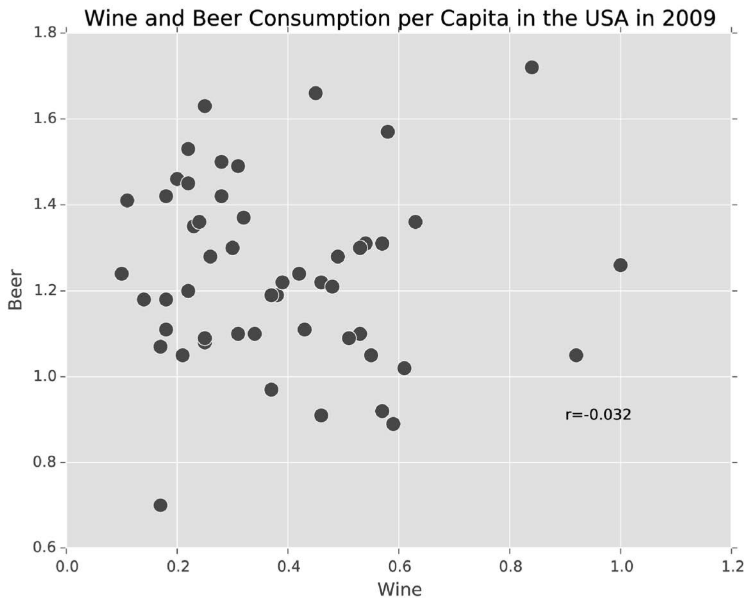
```
⇒              Beer      Wine   Spirits
⇒ Beer     1.000000 -0.031560  0.452279
⇒ Wine    -0.031560  1.000000  0.599791
⇒ Spirits  0.452279  0.599791  1.000000
```

相关性具有极高的*p*值，这为我们的假设提供了致命一击：

```
pearsonr(alco2009["Wine"], alco2009["Beer"])
```

```
⇒ (-0.031560488300856844, 0.82598481310787297)
```

下面的散点图解释了原因：这些点差不多散布在整幅图上，没有特定的顺序或规律。



轮到你了

将本章称为“概率与统计”其实是一种比较鲁莽的做法。即使是单独介绍概率论也需要大量的篇幅，并不是几页就能写清楚的，更何况还有统计方面的内容。无论是概率论还是统计，如果你是第一次接触相关内容，而又想要成为一名数据科学家，那我不得不说，你还需要完成大量的阅读。《统计学习导论：基于R应用》是一个很好的起点，尽管书中使用的R语言可能会使你将其与Python相混淆。但即便你在统计方面是一个不折不扣的新手，仍然可以做一些有趣的项目。让Python伴你前行！

- 21世纪标准普尔500指数*

编写一个程序，给出21世纪标准普尔500指数收盘价的一些基本统计量：平均值、标准差、偏斜度以及收盘价和交易量之间的相关性。为了确定得出的相关性是否可靠，可以从Yahoo! Finance[1]下载历史价格。应注意，21世纪是从2001年1月1日开始的。

- 营养物质网络***

美国农业部（USDA）营养数据库[2]包含大约9000种食物和150种营养成分的信息。当两种营养成分在所有食品中的量具有较强的且稳定的相关性时（相关度大于0.7，p值小于0.01），我们就假定这两种营养成分是相似的。编写一个程序，使用文件NUT_DATA.txt中的营养数据来构建相似营养成分的网络（你可能需要回顾第40单元）。网络中的每种营养成分是一个节点，当两种营养成分相似时，就将对应的两个节点相连。

网络是否存在社区结构？如果存在，在一起的是什么营养成分？

① finance.yahoo.com/q/hp?s=^GSP+Historical+Prices

② www.ars.usda.gov/Services/docs.htm?docid=25700 (document SR28)

在我的迪尔伯恩农场，所有工作都是用机器完成的。

——美国实业家亨利·福特

第 10 章 机器学习

机器学习是一个致力于研究并构建算法的研究领域，这些算法从实验数据中进行学习和预测。机器学习可以分为两大类：监督学习和无监督学习。

监督学习尝试从具有标记的训练数据集中推断出预测函数，其中训练数据集的每个观测样本属于哪一类是已知的（分类结果实际上也是数据集的一部分）。本章中我们将学习线性回归（包括第49单元中的逻辑回归）以及随机决策森林（第51单元）。很遗憾，由于篇幅所限，本章并未包含朴素贝叶斯分类、支持矢量机、线性判别分析和神经网络等内容。

无监督学习尝试在没有标记的数据中找出隐藏的结构。最流行的一些无监督技术包括k均值聚类（第50单元）和社区检测（第40单元第2小节）。分层聚类和主成分分析也是无监督学习的算法，但限于篇幅，本书并不包含相关内容。

这两种类型的机器学习工具都可用于探索性和预测性数据分析。在`SciKit-Learn`模块及其子模块中，可以找到相关工具的Python实现。如果你想实现的功能是对未出现的事物进行预测，而不是对已出现的事物进行解释，那首先要做的就是设置一个预测实验。

第 48 单元 设计预测实验

数据的预测分析绝对是货真价实的科学实验，必须按照严谨的科学实验的方式来组织。数据模型的预测功能不能只是嘴上说说，对其预测能力的评估和验证是实验的重要部分。

请按照以下四个步骤，完成模型的建立、评估和验证。

(1) 将输入数据分成训练集和测试集（建议划分比例为70：30）。然后将测试数据放在一旁，切勿将其用于准备数据模型。

(2) 仅使用训练数据构建数据模型。

(3) 将新模型应用于测试数据。

(4) 使用混淆矩阵或其他质量保证工具评估模型质量。如果模型通过测试，则结束，否则重复以上三个步骤直到模型通过测试。

二元混淆矩阵是一个两行两列的表，用于评估二元预测模型（预测某些属性是否成立的模型）的准确性，如下表所示。

表7　二元混淆矩阵

分类结果		
阳	阴	真实值
真阳（TP）	假阴（FN）	阳
假阳（FP）	真阴（TN）	阴

假设测试集中的每一项是否具有预测属性是已知的，我们使用模型来预测每个项目的属性。（显然，这个假设仅适用于监督学习的模型！）TP是指模型正确地预测了属性为存在（真阳）的项目数；TN是指模型正确地预测了属性为不存在（真阴）的项目数；FP是指模型错误地预测了属性为存在（假阳）的项目数；FN是指模型错误地预测了属性为不存在（假阴）的项目数。

其他机器学习技术



其他监督和无监督机器学习技术包括朴素贝叶斯分类、支持矢量机（SVN）、线性判别分析（LDA）和神经网络。它们中的一些已经包含在SciKit-Learn模块中了。

对混淆矩阵进行归纳可以得出定量的评价指标。

- **准确度**是正确分类项目的比例：

$$\text{准确度} = \frac{TP + TN}{TP + TN + FP + FN}$$

具有较高的准确性是预测模型的最低要求，如果不能保证，模型就不是准确的！

- **精确度**是所有阳性分类中真阳所占的比例：

$$\text{精确度} = \frac{TP}{TP + FP}$$

- **灵敏度**（**或召回**）是在所有阳真实值中真阳所占的比例：

$$灵敏度=\frac{TP}{TP+FN}$$

灵敏度给出了模型识别观测属性的能力好坏。如果真阳是比较罕见的（例如一般人群中的癌症病例），模型就必须足够敏感才行。

- **特异性**是在所有阴真实值中真阴所占的比例：

$$特异性=\frac{TN}{TN+FP}$$

较高的特异性意味着模型能很好地捕获该属性的缺失。

许多统计模型具有高灵敏度、低特异性，或低灵敏度、高特异性，具体是哪一种取决于模型参数。参数选择的依据是：哪个度量对你而言更重要，就选择相应的参数。如果预测模型的特异性和灵敏度都很低，则可以将其转化成为一个很好的预测因子。

如果预测值不是二元的（例如分类或连续值），则必须使用其他质量控制工具。在本章的后续部分中，我们将学习部分这样的工具。

第 49 单元

线性回归拟合

线性回归是一种预测统计模型，其目的是使用线性模型来解释变量的全部或部分变化规律。这是一种监督建模技术：在预测之前，必须对模型进行训练（“拟合”）。

普通最小二乘回归

普通最小二乘（OLS）回归将自变量（预测因子）和因变量（预测值）联系在一起。该模型将预测值$reg(x_i)$视为预测因子x_i的线性组合。真实值y_i和预测值之间的差异称为残差。在最佳拟合的情况下，所有残差都为零。可能的加权（权重$w_i>0$）残差的平方和（SSR）给出了拟合的质量。训练模型的过程就是最小化SSR的过程：

$$SSR=\sum_i^N (y_i-reg(x_i))^2 w_i^2 \to min$$

另一种拟合质量的度量是模型**得分**，也称为R^2。得分$0\leqslant R^2\leqslant 1$表示拟合模型（和理想情况）的差异大小。在最佳拟合的情况下，$R^2=1$。对于非常差的拟合，$R^2\approx 0$。

sklearn.linear_model模块中的构造函数LinearRegression()可以创建一个OLS回归对象。

fit()函数使用1×n的预测因子矩阵。如果模型具有一个自变量，且预测因子是矢量的形式，则使用numpy.newaxis对象的切片操作创建另一个维度。将因变量的真值作为一维矢量传递给fit()函数。（如果想对多个属性进行预测，则必须构建和拟合多个模型。）

拟合完成后，可以使用回归对象来计算预测值（函数predict()）和得出拟合分数（函数score()）。属性coef_和intercept_分别包含回归系数和拟合后的截距值。

以下示例使用Yahoo! Finance[①]的标准普尔500收盘价记录来建立、拟合和评估线性回归模型。假定数据之前被保存为文件sapXXI.csv。

首先，导入所有必要的模块并加载标准普尔500数据：

sap-linregr.py

```
import numpy, pandas as pd
import matplotlib, matplotlib.pyplot as plt
import sklearn.linear_model as lm

# 获取数据
sap = pd.read_csv("sapXXI.csv").set_index("Date")
```

可以看出数据的行为通常是非线性的，但是从2009年1月1日开始有一个很好的、几乎线性的片段，并延伸到数据集的结尾。下面仅使用该片段进行模型拟合。可惜的是，SciKit-Learn不直接支持日期这种数据格式。因此，必须将其转换为序数，即天数，然后才能创建、拟合和评估线性回归模型，并算出标准普尔500指数的预测值。模型得分是0.95，这个值是可以接受的！

sap-linregr.py

```
# 选取一段“线性变化”的数据
sap.index = pd.to_datetime(sap.index)
sap_linear = sap.ix[sap.index > pd.to_datetime('2009-01-01')]

# 准备模型并拟合
olm = lm.LinearRegression()
X = numpy.array([x.toordinal() for x in sap_linear.index])[:, numpy.newaxis]
y = sap_linear['Close']
olm.fit(X, y)

# 计算预测值
yp = [olm.predict(x.toordinal())[0] for x in sap_linear.index]

# 评估模型
olm_score = olm.score(X, y)
```

最后，代码会将原始数据集、预测直线，甚至模型分数都绘制出来。结果如下面代码后的图所示。

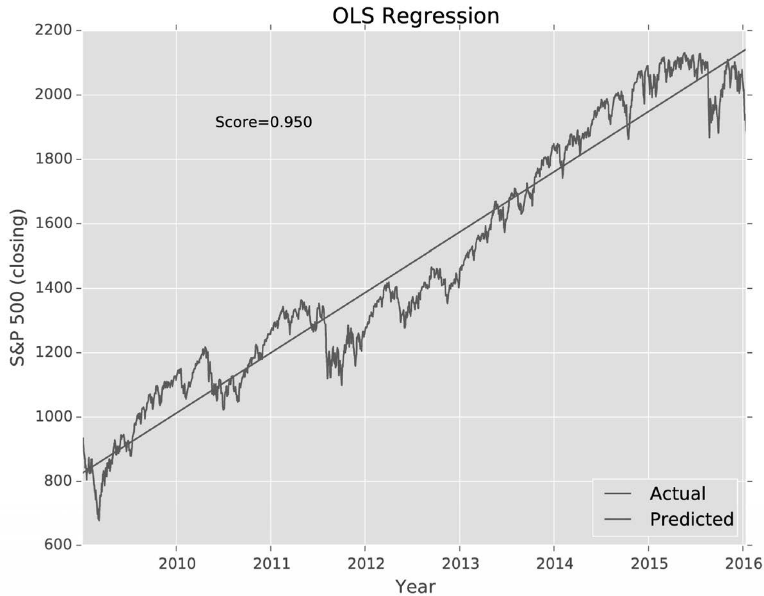
① finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices

sap-linregr.py

```
# 选择一种优美的绘图样式
matplotlib.style.use("ggplot")

# 绘制两个数据集
plt.plot(sap_linear.index, y)
plt.plot(sap_linear.index, yp)

# 加入图饰
plt.title("OLS Regression")
plt.xlabel("Year")
plt.ylabel("S&P 500 (closing)")
plt.legend(["Actual", "Predicted"], loc="lower right")
plt.annotate("Score=%.3f" % olm_score,
             xy=(pd.to_datetime('2010-06-01'), 1900))
plt.savefig("../images/sap-linregr.pdf")
```



有一个不尽如人意的地方，那就是`SciKit-Learn`没有计算拟合的p值，因此不能确定拟合是否有效。

如果要将一些非线性预测因子（例如平方、平方根和对数）甚至原始预测因子的组合添加到模型中，只需将这些函数和组合看作新的自变量即可。

脊回归

如果存在两个或更多个预测因子是高度相关的（所谓的**共线性**的情况），则拟合OLS回归可

能产生非常大的系数。你可以通过施加惩罚因子来限制回归系数的这种不可控增长。脊回归就是这样一种广义线性模型，它使用复杂度参数α来实现对系数的抑制。该过程称为模型的**正则化**：

$$\mathrm{SSR_{gen}} = \sum_i^N (y_i - \mathrm{reg}(x_i))^2 w_i^2 + \alpha \sum_i^N \mathrm{coeff}_i^2 \to \min$$

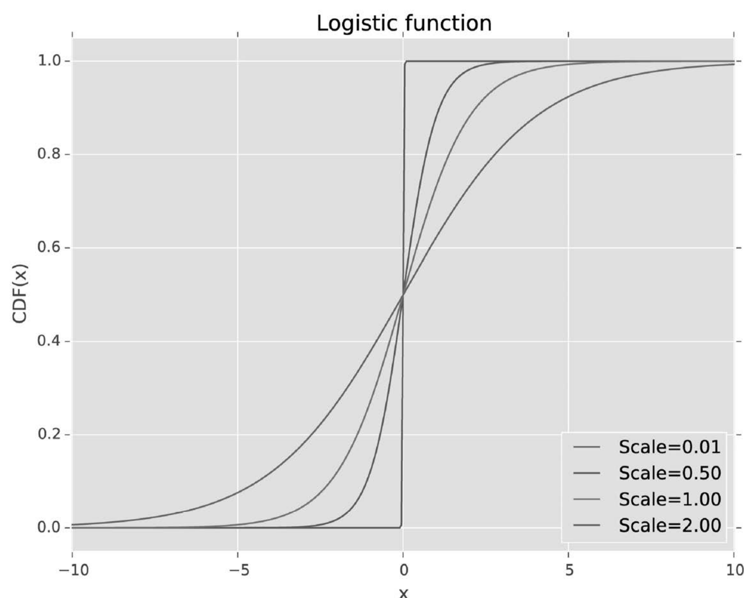
当α=0时，脊回归退化为OLS回归。α越大，惩罚越大；模型拟合会得出较小的系数，但这样也有可能使模型变得更糟。

函数`Ridge()`创建一个脊回归对象，它以α为参数。创建对象后，可以像使用OLS回归一样使用它：

```
regr = lm.Ridge(alpha=.5)
regr.fit(X, y)
«...»
```

逻辑回归

虽然名称中有“回归”，但逻辑回归并不是回归，实际上它是一种二元分类工具。它使用广义逻辑函数（逻辑函数的扩展，也称为s曲线或sigmoid函数，如下图所示）。广义逻辑函数的特征包括上下渐近线、sigmoid函数中点的x值和曲线的陡度。



函数`LogisticRegression()`创建一个逻辑回归对象的实例。它需要几个可选参数，其中最重要的是参数`C`。

参数C是正则化参数的倒数（即脊回归的α的倒数）。通常，为了使分类的结果有意义，其取值至少为20。默认值C=1.0在许多实际情况下是不合理的。

因变量y可以是整数、布尔值或字符串。

sklearn.linear_model通过predict()函数实现分类。与线性回归模型（OLS和脊回归）不同，逻辑回归模型的预测结果通常比模型系数coef_和截距intercept_更有价值。

下面的例子将使用43名学生的计算机科学导论课程的匿名测验成绩（可在文档grade.csv中找到）来阐述逻辑回归。通过分析，研究是否能使用前两次的测验结果（共十次测试）来预测学生的最终成绩，或至少预测出成绩是否达标（“C”或以上）：

logit-example.py

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import sklearn.linear_model as lm

# 初始化回归工具
clf = lm.LogisticRegression(C=10.0)

# 读取数据，使用字母给出成绩的量化等级
grades = pd.read_table("grades.csv")
labels = ('F', 'D', 'C', 'B', 'A')
grades["Letter"] = pd.cut(grades["Final score"], [0, 60, 70, 80, 90, 100],
                          labels=labels)
X = grades[["Quiz 1", "Quiz 2"]]

# 拟合，并给出模型得分和混淆矩阵
clf.fit(X, grades["Letter"])
print("Score=%.3f" % clf.score(X, grades["Letter"]))
cm = confusion_matrix(clf.predict(X), grades["Letter"])
print(pd.DataFrame(cm, columns=labels, index=labels))
```

```
⇒ Score=0.535
⇒    F   D  C  B  A
⇒ F  0   0  0  0  0
⇒ D  2  16  6  4  1
⇒ C  0   1  6  2  2
⇒ B  0   0  0  1  2
⇒ A  0   0  0  0  0
```

我们使用sklearn.metrics模块中的confusion_matrix()函数来计算混淆矩阵（参见表7）。模型得分看起来不太准确：该模型能够准确预测的成绩只占所有成绩的54%左右。然而，混淆矩阵的主对角线（正确预测）及其邻域几乎包含了所有的非零条目。这意味着模型的预测要么是准确的，要么是结果具有±1个字母的偏差。对大多数实际应用而言，这种“扩展”精度（84%）实际上已经足够了。

第 50 单元

用 k 均值聚类实现数据分组

聚类是一种无监督的机器学习技术。你不需要（也不能）对模型进行训练。

聚类的目标是将样本（每个样本用一个*n*维实数矢量表示）划分到具有较好内部邻近度的不相交紧凑组中。要实现聚类，矢量维度必须具有合理的、可兼容的取值范围。如果某一维的取值范围比其他维的取值范围大得多或小得多，就应该在聚类之前对“太大”或“太小”的变量进行缩放。

k均值聚类按照下面的算法将样本聚合成*k*个类（该算法也因此得名）。

(1) 随机选择*k*个矢量作为初始质心（矢量不需要是数据集中的样本）。

(2) 将每个样品分配给它最接近的质心。

(3) 重新计算质心位置。

(4) 重复步骤(2)和步骤(3)，直到质心不再移动。

`sklearn.cluster`模块通过`KMeans`对象实现k均值算法，该对象具有用于实际聚类的`fit()`函数、用于将新样本分配给预先计算好的聚类的`predict()`函数，以及用于同时完成聚类和标记的`fit_predict()`函数。

模块`sklearn.preprocessing`具有实现变量缩放的`scale()`函数。该函数将每一维变量减去其最小值，再除以其取值范围，从而将每一维变量都映射到[0…1]的区间内。

`cluster_centers_`和`labels_`属性包含描述最终质心的矢量，以及分配给每个样本类的数字标签。其中，标签并不反映类的目的和组成。如果要为类分配有意义的标签，可以执行以下任意一种操作。

- 使用人的智能（查看样本并提供一个通用标签）。
- 使用众包（例如亚马逊的MTurk工作人员）。
- 从数据生成标签（例如将具有主导地位的样本的一个属性指定为类标签）。

在第47单元第2小节中，我们试图对美国在2009年的葡萄酒和啤酒消费情况进行分析，得出的结论是两者并不是线性相关的。现在让我们通过聚类给这些“高贵”饮料另一个机会。分析的流程及结果如下：

clusters.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd
import sklearn.cluster, sklearn.preprocessing

# NIAAA frame已完成了pickle操作
alco2009 = pickle.load(open("alco2009.pickle", "rb"))
# 州名缩写
states = pd.read_csv("states.csv",
                     names=("State", "Standard", "Postal", "Capital"))
columns = ["Wine", "Beer"]
# 初始化聚类对象，模型拟合
kmeans = sklearn.cluster.KMeans(n_clusters=9)
kmeans.fit(alco2009[columns])
alco2009["Clusters"] = kmeans.labels_
centers = pd.DataFrame(kmeans.cluster_centers_, columns=columns)

# 选取一种美观的显示样式
matplotlib.style.use("ggplot")

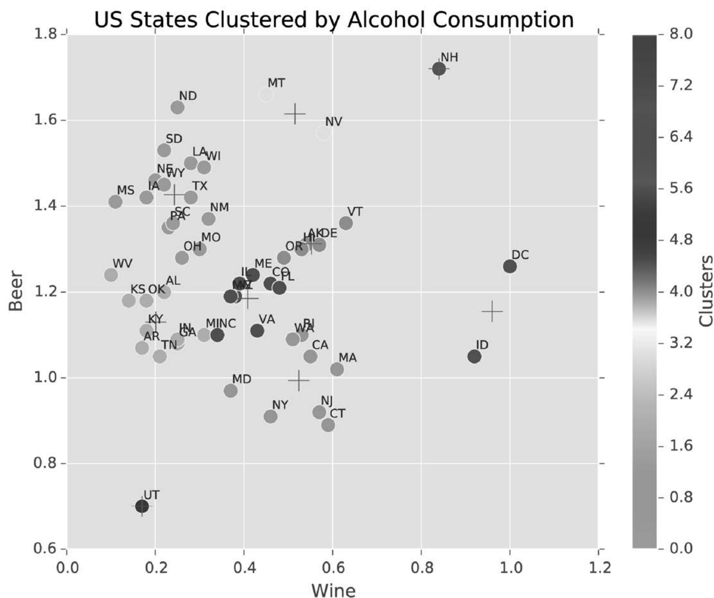
# 绘制州和聚类中心
ax = alco2009.plot.scatter(columns[0], columns[1], c="Clusters",
                           cmap=plt.cm.Accent, s=100)
centers.plot.scatter(columns[0], columns[1], color="red", marker="+",
                     s=200, ax=ax)

# 将州的缩写作为注释加入图中
def add_abbr(state):
    _ = ax.annotate(state["Postal"], state[columns], xytext=(1, 5),
                    textcoords="offset points", size=8,
                    color="darkslategrey")
alco2009withStates = pd.concat([alco2009, states.set_index("State")],
                               axis=1)
alco2009withStates.apply(add_abbr, axis=1)

# 加入标题，并保存绘图结果
plt.title("US States Clustered by Alcohol Consumption")
plt.savefig("../images/clusters.pdf")
```

请注意，除非传递了`n_clusters`参数，否则`KMeans()`函数总是生成八个类。如何选择类的数量取决于你和你的直觉。

我们将原始数据（实心圆圈，用州名缩写标记）和类的质心（十字符号）绘制在同一个图表中，如下所示：



KMeans()函数在类的识别方面做了很好的工作(比如葡萄酒和啤酒的饮用量处于中等水平的东北部)。标签的放置存在许多不足之处，但该主题不在本书的讨论范围之内。

Voronoi单元



k均值聚类算法将自变量空间分为Voronoi单元——由与一个种子（数据点）的距离小于任何其他种子的点组成的区域。

第 51 单元

在随机决策森林中生存

决策树是一种监督型的机器学习工具。它使用树形图这样的结构，其中每个节点包含对特定数据集属性的测试，与节点相关联的分支对应测试结果。树在使用前必须经过训练。训练的内容包括用树来表示各种预测因子以及相应的标签（特征），并相应地调整节点测试条件。（当然，这些训练工作是不需要手动完成的！）

随机决策森林回归使用一些（一整套）针对数据集各种子样本的分类决策树，并通过求预测结果的平均值以提高准确性。模块sklearn.ensemble提供了构造器RandomForestRegressor()。回归对象具有fit()、predict()等函数，它们与目前为止你在本章中看到的其他回归具有相同

的语法和语义。

下面我们使用随机决策森林来分析波士顿地区的特征价格统计数据[①]，该数据是D. Harrison和D. Rubinfeld在1978年首先公布的。该数据集提供了506个房价中值（实验中的标签为mv）和14个其他变量（预测因子）。

rfr.py

```
from sklearn.ensemble import RandomForestRegressor
import pandas as pd, numpy.random as rnd
import matplotlib, matplotlib.pyplot as plt

# 读取数据，准备两个随机的互补数据集
hed = pd.read_csv('Hedonic.csv')
selection = rnd.binomial(1, 0.7, size=len(hed)).astype(bool)
training = hed[selection]
testing = hed[-selection]

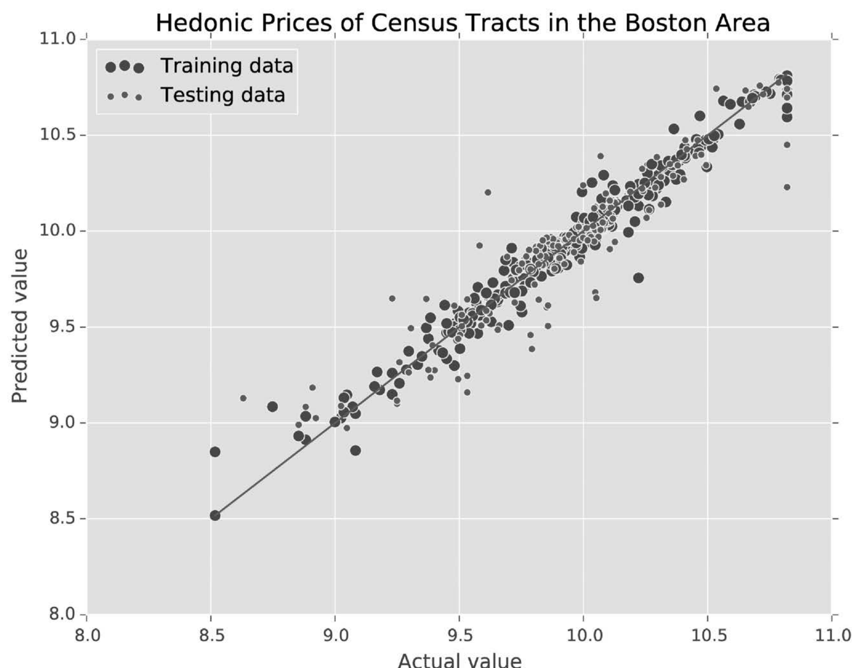
# 创建回归和预测集
rfr = RandomForestRegressor()
predictors_tra = training.ix[:, "crim" : "lstat"]
predictors_tst = testing.ix[:, "crim" : "lstat"]

# 模型拟合
feature = "mv"
❶ rfr.fit(predictors_tra, training[feature])
# 选择一种美观的样式
matplotlib.style.use("ggplot")

# 绘制预测结果
plt.scatter(training[feature], rfr.predict(predictors_tra), c="green",
❷             s=50)
❸ plt.scatter(testing[feature], rfr.predict(predictors_tst), c="red")
plt.legend(["Training data", "Testing data"], loc="upper left")
plt.plot(training[feature], training[feature], c="blue")
plt.title("Hedonic Prices of Census Tracts in the Boston Area")
plt.xlabel("Actual value")
plt.ylabel("Predicted value")
plt.savefig("../images/rfr.pdf")
```

我们使用训练数据集（从完整数据集中随机选择的）来拟合预测因子❶，然后在训练集（❷）以及未用于拟合的测试集（❸）上进行测试。由于预测特征mv不是离散的，这不利于使用混淆矩阵来评估模型质量。因此，我们采取了图形化的方式，结果表明两个数据集的预测质量至少是合理的。模型看起来相当准确，没有过拟合现象：

① rcom.univie.ac.at/mirrors/lib.stat.cmu.edu/datasets/boston



轮到你了

本章只是机器学习的冰山一角。（根据“传统”的科学，典型的冰山有三分之二被隐藏在水下。）即便如此，你还是拥有足够强大的工具和知识来完成监督和无监督的数据处理。你可以设置描述性（回归和分类）和预测性的数据科学实验，并在一定程度上评估其有效性，得出不凡的结论。

祝贺你！现在你几乎已经是一名数据科学家了。（但你仍然需要完成剩余的项目。）

❑ MOSN分类*

编写一个程序，通过注册用户数量和全球Alexa页面排名[1]，对大量的在线社交网站进行分类。由于站点的排名及其大小差异很大，所以应使用对数尺度来进行聚类和结果呈现。

❑ 分片线性的标准普尔500**

编写一个程序，从Yahoo! Finance[2]获得21世纪标准普尔500的收盘价的历史记录，并将每

① en.wikipedia.org/wiki/List_of_social_networking_websites

② finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices

年价值的观测值表示为OLS回归模型。如果模型得分不够（0.95或以下），就将时间间隔分成两半，并采用递归的处理方式，直到OLS模型得分超过0.95或时间间隔短于一周，然后将标准普尔500价格的原始值和OLS模型的预测值绘制在一幅图中。

- **地铁预测*****

编写一个程序，预测一个城市是否拥有地铁系统。程序可能需要用到人口总量、人口密度、预算大小、天气状况、所得税水平和其他变量。其中有些可能很容易在网上查到，而有些则不是。使用逻辑回归和随机决策森林，并选出性能最佳的方法。

如果你找到了一本书里特别吸引你的段落，那就通过大声朗读开始学习吧。

——北美作家Grenville Kleiser

附录 1 扩展阅读

当你读到这里的时候，应该已经发现本书并未涉及多么高深的内容，本书假设你基本上知道应该做什么，只是不确定应该怎么做。我将在这里列出几本优秀的图书，供大家深入学习相关主题，当然，这些书中的内容难免存在一定程度的重复。

如果你不熟悉数据科学，但知道R语言或者至少不介意学习R语言，那么你可以参考《统计学习导论：基于R应用》和《数据科学：理论、方法与R语言实践》。前者是具有实用编程元素的统计学书，后者则是具有统计学元素的实用书，它们是一个很好的组合。另一本书*The Elements of Data Analytic Style* [Lee15]涉及不同的数据模型类型、报告撰写、创建支撑图片和编写可复现的代码。

《利用Python进行数据分析》是pandas的创作者Wes McKinney的经典pandas图书，涵盖了你想知道的关于pandas和numpy的所有内容，包括金融时间序列分析。这本书对很多案例进行了非常详细的分析。

《Python自然语言处理》既是一本Python教程，也是一个完整的NLP解决方案。这本书假定你并不知道Python，它不仅介绍了文本归一化和文字计数，还介绍了文本分类、句子结构分析和语义分析。你可以免费获取官方在线版本[1]！

社交网站是一个大型的迅猛扩展的原始数据仓库。《社交网站的数据挖掘与分析》仔细分析了应用程序的编程接口（API），使得你可以用Unix风格的邮箱、Twitter、LinkedIn、Google Buzz和Facebook。它很好地概述了最重要的自然语言处理任务。可惜的是，尽管这本书是近几年出版的，但它的大部分内容已经过时了：一些API已经改变，一些社交网络项目（比如Google Buzz）已经被终止了。

① www.nltk.org/book

《MySQL必知必会》正如它所声称的，是关于如何建立、维护和操作关系数据库的全面的速成课。这本书没有涉及Python或任何其他语言的API。

在撰写本文时，还没有关于网络分析的Python图书。*Network Analysis: Methodological Foundations* [BE05]这本书并不适用于计算机程序员，实际上这是一本理论性非常强的书。《社交网络分析》对于不喜欢定理、证明和冗长公式的从业者来说更为合适。虽然使用了《社交网络分析》这个书名，但这本书的内容不仅限于社交网络，还对网络进行了很好的介绍。

最后，《数据科学入门》是本书的扩展。它扩展了统计学和机器学习的内容，是最适合接下来阅读的一本书。

在许多情况下，我们推断不出问题的正确解决方案，因为我们缺少数据。

——北美伦理学研究员Durant Drake

附录 2 单星项目的解决方案

此附录挑选了一些（单星）项目，给出了参考的解决方案。这些解决方案使用了最具Python风格的方式。如果你的解决方案与给出的不一致，也没有必要纠结！编程问题的解决方案本来就不止一种，这就如同有多种描写爱情和死亡的方法一样。

- Hello, World!

  编写一个在Python命令行输出“Hello, World!”的程序（第1章的问题）。

solution-hello.py

```
# 尊重传统
print("Hello, World!")
```

- 词频计数器

  编写一个程序，用于下载用户请求的网页，并给出网页中使用频率最高的十个词，所有词不区分大小写。出于练习的目的，可以简单地假设一个词由正则表达式`r"\w+"`确定（第2章的问题）。

solution-counter.py

```
import urllib.request, re
from collections import Counter

# 建立与用户以及与网络的会话
url = input("Enter the URL: ")
try:
    page = urllib.request.urlopen(url)
except:
    print("Cannot open %s" % url)
    quit()

# 读取页面，并完成部分规范化操作
```

```
doc = page.read().decode().lower()

# 将文本切分为词语
words = re.findall(r"\w+", doc)

# 构建计数器并给出结果
print(Counter(words).most_common(10))
```

- 失效链接检测器

编写一个程序，基于一个给定网页的URL，报出页面中失效链接的名称和地址。出于练习的目的，约定当使用`urllib.request.urlopen()`打开链接失败时，则认为链接失效（第3章的问题）。

solution-broken_link.py

```
import urllib.request, urllib.parse
import bs4 as BeautifulSoup

# 建立与用户以及与网络的会话
base = input("Enter the URL: ")
try:
    page = urllib.request.urlopen(base)
except:
    print("Cannot open %s" % base)
    quit()

# 准备soup
soup = BeautifulSoup.BeautifulSoup(page)

# 提取链接，并用(名称,网址)的元组表示
links = [(link.string, link["href"])
         for link in soup.find_all("a")
         if link.has_attr("href")]

# 尝试打开每个链接
broken = False
for name, url in links:
    # 将base和链接目标组合在一起
    dest = urllib.parse.urljoin(base, url)
    try:
        page = urllib.request.urlopen(dest)
        page.close()
    except:
        print("Link \"%s\" to \"%s\" is probably broken." % (name, dest))
        broken = True

# 显示好消息!
if not broken:
    print("Page %s does not seem to have broken links." % base)
```

- MySQL文件索引器

编写一个Python程序，对于给定文件中的每个单词，记录如下信息到MySQL数据库中：单词本身（不是词干！）、单词在文件中的序数（从1开始），以及单词的词性标记。使用NLTK WordPunct-Tokenizer（参考第16单元第2小节）来识别单词。假设这些单词比较短，数据类型可以使用MySQL的TINYTEXT。设计数据库模式，创建所有必需的表，并在正式开始编写Python代码之前，通过命令行来试用一下设计出来的表（第4章的问题）。

该解决方案由两个文件组成：一个是设置表的MySQL脚本，另一个是执行索引的Python程序。

solution-mysql_indexer.sql

```
CREATE TABLE IF NOT EXISTS indexer(id INT PRIMARY KEY AUTO_INCREMENT,
                                   ts TIMESTAMP,
                                   word TINYTEXT,
                                   position INT,
                                   pos VARCHAR(8));
```

solution-mysql-indexer.py

```
import nltk, pymysql

infilename = input("Enter the name of the file to index: ")

# 结合你的MySQL服务器的设置改变本行
conn = pymysql.connect(user="dsuser", passwd="badpassw0rd", db="dsbd")
cur = conn.cursor()

QUERY = "INSERT INTO indexer (word,position,pos) VALUES "
wpt = nltk.WordPunctTokenizer()

offset = 1
with open(infilename) as infile:
    # 使用增量方式处理文本，每次处理一行
    # 不论如何，一个词不可能跨行分布
    for text in infile:
        # 分词并加入POS标签
        pieces = enumerate(nltk.pos_tag(wpt.tokenize(text)))

        # 创建一个查询命令；别忘了要避开待查询的词！
        words = ["(\"%s\",%d,\"%s\")" % (conn.escape_string(w),
                                         i + offset,
                                         conn.escape_string(pos))
                 for (i, (w, pos)) in pieces]

        # 执行查询命令
        if words:
            cur.execute(QUERY + ','.join(words))

            # 移动词的位置指针
            offset += len(words)
```

```
# 提交更改
conn.commit()
conn.close()
```

❑ 数组微分器

（函数的）部分和近似等价于（函数的）积分。事实上，微积分理论就把积分定义为无穷小元素的无限求和。（函数的）部分差$arr_{i+1}-arr_i$近似等价于（函数的）导数。numpy没有提供用于计算数组部分差的工具。请编写一个程序，对于一个给定的数组arr，计算数组项的部分差。在本练习中假设数组为数值数组（第5章的问题）。

solution-difference.py

```
import numpy as np

# 用于测试的合成数据
array = np.random.binomial(5, 0.5, size=100)

# 计算数组部分差：切片&广播!
diff = array[1:] - array[:-1]
```

❑ 猞猁的捕获量

写一个程序，使用每年加拿大猞猁的捕获量[①]给出每十年的猞猁捕获总量，并将结果按逆序排列（最“有效益的”十年排在前面）。如果cache目录中不存在数据文件，则程序自动将数据下载到cache目录中。如果cache目录不存在，则程序将自动创建该目录。程序将计算结果保存到doc目录下的CSV文件中。如果doc目录不存在，则程序将自动创建该目录（第6章的问题）。

solution-lynx.py

```
import os, pandas as pd
import urllib.request

# 一些“常量”
SRC_HOST = "https://vincentarelbundock.github.io"
FILE = "/lynx.csv"
SRC_NAME = SRC_HOST + "/Rdatasets/csv/datasets" + FILE
CACHE = "cache"
DOC = "doc"

# 在需要时创建目录
if not os.path.isdir(CACHE):
    os.mkdir(CACHE)
if not os.path.isdir(DOC):
    os.mkdir(DOC)

# 检查文件是否缓存；如未缓存就进行缓存处理
```

① vincentarelbundock.github.io/Rdatasets/csv/datasets/lynx.csv

```
if not os.path.isfile(CACHE + FILE):
    try:
        src = urllib.request.urlopen(SRC_NAME)
        lynx = pd.read_csv(src)
    except:
        print("Cannot access %f." % SRC_NAME)
        quit()
    # 创建数据frame
    lynx.to_csv(CACHE + FILE)
else:
    lynx = pd.read_csv(CACHE + FILE)

# 加入“decade”列
lynx["decade"] = (lynx['time'] / 10).round() * 10

# 求和并排序
by_decade = lynx.groupby("decade").sum()
by_decade = by_decade.sort_values(by="lynx", ascending=False)

# 保存结果
by_decade["lynx"].to_csv(DOC + FILE)
```

❑ 中心性的相关问题

从斯坦福大网络数据集[①]中下载Epinions.com网站用户的社交网络图，并提取出规模排名第十的社区。编写一个程序，计算并显示第7章中提到的所有网络中心性度量之间的两两相关性；加入集聚系数可以使问题变得更加有趣（第7章的问题）。建议你使用pandas的frame来存储所有中心性。你可能需要阅读第47单元第2小节来了解如何使用pandas计算相关性。

是否存在强相关的一对中心性？

solution-centrality.py

```
import networkx as nx, community
import pandas as pd

# 导入network模块
G = nx.read_adjlist(open("soc-Epinions1.txt", "rb"))

# 提取社区结构并用series保存
partition = pd.Series(community.best_partition(G))

# 找出10个最大社区的索引
top10 = partition.value_counts().index[9]

# 提取10大社区
# 注意节点的标签是字符串!
subgraph = partition[partition == top10].index.values.astype('str')
F = G.subgraph(subgraph)
```

① snap.stanford.edu/data/soc-Epinions1.html

```
# 计算网络的度量
df = pd.DataFrame()
df["degree"] = pd.Series(nx.degree_centrality(F))

df["closeness"] = pd.Series(nx.closeness_centrality(F))
df["betweenness"] = pd.Series(nx.betweenness_centrality(F))
df["eigenvector"] = pd.Series(nx.eigenvector_centrality(F))
df["clustering"] = pd.Series(nx.clustering(F))

# 计算相关性
print(df.corr())
```

```
⇛              degree  closeness  betweenness  eigenvector  clustering
⇛ degree      1.000000   0.247377     0.871812     0.738836    0.100259
⇛ closeness   0.247377   1.000000     0.169449     0.547228    0.024052
⇛ betweenness 0.871812   0.169449     1.000000     0.527290   -0.015759
⇛ eigenvector 0.738836   0.547228     0.527290     1.000000    0.143070
⇛ clustering  0.100259   0.024052    -0.015759     0.143070    1.000000
```

度中心性与中介中间性以及特征矢量中心性呈强的线性相关关系。

- 美国派

编写一个程序，将美国的所有州按首字母分组，并用一个饼图显示分组结果或保存为PDF文件。为了完成该项目，你需要用到州的名称或其缩写的列表，这个列表可以从命名网站获取[①]（第8章的问题）。

solution-states-pie.py

```
import pandas as pd
import matplotlib, matplotlib.pyplot as plt

def initial(word):
    return word[0]

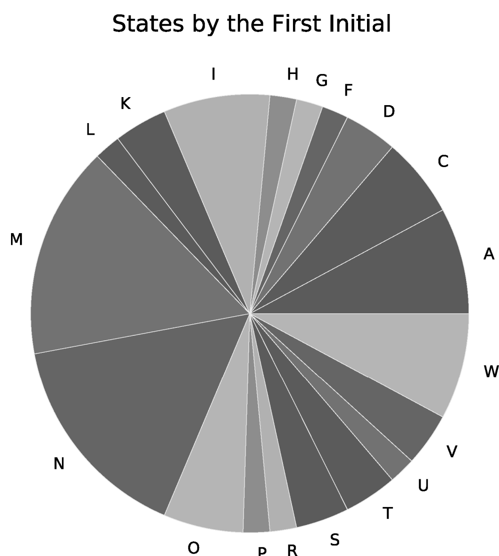
# 读取州名（可以使用任何你喜欢的数据源！）
states = pd.read_csv("states.csv",
                     names=("State", "Standard", "Postal", "Capital"))
# 选取一种优美的绘图样式
matplotlib.style.use("ggplot")

# 绘图
plt.axes(aspect=1)
states.set_index('Postal').groupby(initial).count()['Standard'].plot.pie()
plt.title("States by the First Initial")
plt.ylabel("")

plt.savefig("../images/states-pie.pdf")
```

① www.stateabbreviations.us

得到的饼图如下所示：



❑ 21世纪标准普尔500指数

编写一个程序，给出21世纪标准普尔500指数收盘价的一些基本统计量：平均值、标准差、偏斜度以及收盘价和交易量之间的相关性。为了确定得出的相关性是否可靠，可以从Yahoo! Finance[①]下载历史价格。应注意，21世纪是从2001年1月1日开始的（第9章的问题）。

鉴于新下载的数据不同于本示例中使用的数据，你的答案可能也会不同。

solution-sap.py

```
import pandas as pd
from scipy.stats import pearsonr

# 读取数据
sap = pd.read_csv("sapXXI.csv").set_index("Date")

# 计算并给出统计值的报告
print("Mean:", sap["Close"].mean())
print("Standard deviation:", sap["Close"].std())
print("Skewness:", sap["Close"].skew())
print("Correlation:\n", sap[["Close", "Volume"]].corr())
_, p = pearsonr(sap["Close"], sap["Volume"])

print("p-value:", p)
```

```
⇒ Mean: 1326.35890044
⇒ Standard deviation: 332.784759688
⇒ Skewness: 0.858098114571
```

① finance.yahoo.com/q/hp?s=^GSP+Historical+Prices

```
⇛ Correlation:
⇛             Close   Volume
⇛ Close   1.000000 0.103846
⇛ Volume  0.103846 1.000000
⇛ p-value: 1.5301705092e-10
```

结果表明相关性是非常可靠的，但非常微不足道。

- MOSN分类

编写一个程序，通过注册用户数量和全球Alexa页面排名[①]，对大量的在线社交网站进行分类。由于站点的排名及其大小差异很大，所以应使用对数尺度来进行聚类和结果呈现（第10章的问题）。

solution-mosn.py

```
import pandas as pd, numpy as np
import sklearn.cluster, sklearn.preprocessing
import matplotlib, matplotlib.pyplot as plt

# 读取数据
mosn = pd.read_csv('mosn.csv', thousands=',',
                   names=('Name', 'Description', 'Date', 'Registered Users',
                          'Registration', 'Alexa Rank'))
columns = ['Registered Users', 'Alexa Rank']

# 删除有缺失数据和零值的行
good = mosn[np.log(mosn[columns]).notnull().all(axis=1)].copy()

# 聚类
kmeans = sklearn.cluster.KMeans()
kmeans.fit(np.log(good[columns]))
good["Clusters"] = kmeans.labels_

# 找出Facebook
fb = good.set_index('Name').ix['Facebook']['Clusters']

# 选取一种优美的绘图样式
matplotlib.style.use("ggplot")

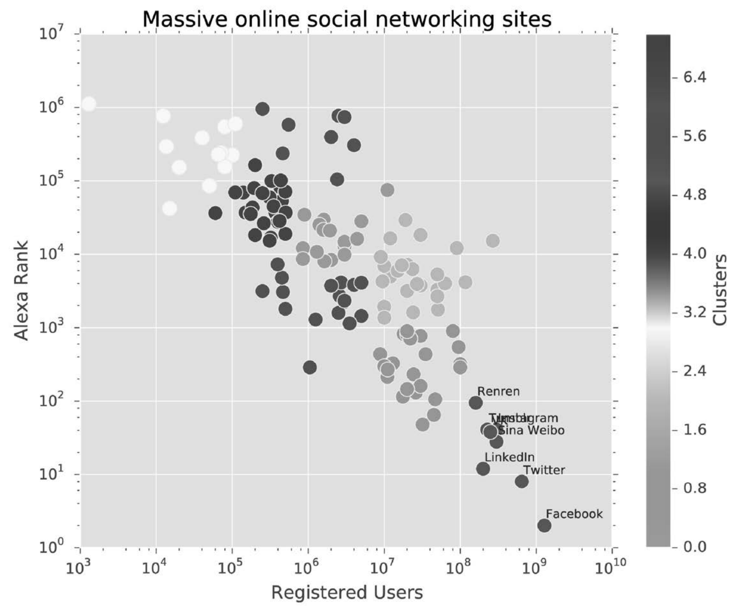
# 显示结果
ax = good.plot.scatter(columns[0], columns[1], c="Clusters",
                       cmap=plt.cm.Accent, s=100)
plt.title("Massive online social networking sites")
plt.xscale("log")
plt.yscale("log")

# 标记出最出名的站点
def add_abbr(site):
    if site['Clusters'] == fb:
        _ = ax.annotate(site["Name"], site[columns], xytext=(1, 5),
```

① en.wikipedia.org/wiki/List_of_social_networking_websites

```
                        textcoords="offset points", size=8,
                        color="darkslategrey")
good.apply(add_abbr, axis=1)

plt.savefig("../images/mosn.png")
```



参考文献

[BE05] Ulrik Brandes and Thomas Erleback. *Network Analysis: Methodological Foundations.* Springer, New York, NY, 2005.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O'Reilly & Associates, Inc., Sebastopol, CA, 2009.

[For05] Ben Forta. *MySQL Crash Course.* Sams Publishing, Indianapolis, IN, 2005.

[Gru15] Joel Grus. *Data Science from Scratch: First Principles with Python.* O'Reilly & Associates, Inc., Sebastopol, CA, 2015.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer, New York, NY, 2013.

[KY07] David Knoke and Song Yang. *Social Network Analysis.* SAGE Publications, Thousand Oaks, CA, 2nd, 2007.

[Lee15] Jeff Leek. *The Elements of Data Analytic Style.* Leanpub, Victoria, BC, Canada, 2015.

[McK12] Wes McKinney. *Python for Data Analysis.* O'Reilly & Associates, Inc., Sebastopol, CA, 2012.

[Rus11] Matthew A. Russell. *Mining the Social Web.* O'Reilly & Associates, Inc., Sebastopol, CA, 2011.

[ZM14] Nina Zumel and John Mount. *Practical Data Science with R.* Manning Publications Co., Greenwich, CT, 2014.

Data Science Essentials in Python

Python数据科学入门

- 不同类型文本数据的获取、清洗、组织和可视化
- 如何用NumPy和pandas模块处理数值数据
- 探索用MySQL和MongoDB配置、填充、查询数据
- 网络创建、度量和分析
- 概率与统计以及机器学习的相关基本概念

“这本书完成了一项非常了不起的工作，它对各种使用Python进行数据整理的活动进行了总结。每个练习都是一项有意思的挑战，求解的过程也非常有趣。毫无疑问，这本书应该出现在每一位有抱负的数据科学家的阅读清单中。”

——Peter Hampton，阿尔斯特大学

“这本书可以让你快速地熟悉数据科学领域中各种常见任务和工具。书中简要介绍了多种用于数据获取、清洗、分析和存储的技术，可以帮助你保持较高的工作效率，减少钻研技术的时间，从而在期望开展的研究上投入更多精力。”

——Jason Montojo，

*Practical Programming: An Introduction to Computer Science Using Python 3*一书合著者

“对于那些对解决问题和数据探索非常好奇且充满热情的人来说，这本书提供了深刻的见解以及起步所需的正确的工具和技术。精心设计的示例和练习使该书更具实用性和可读性。”

——Lokesh Kumar Makani，Skyhigh Networks的CASB专家

The Pragmatic Programmers

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/数据科学/Python

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-47060-7

定价：49.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱: ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks